DTIC FILE COPY

AD-A215 740

(1)

DTIC
ELECTE
DEC 15 1989
S B D

Payload Invariant Control via Neural Networks: Development and Experimental Evaluation

THESIS

Mark Alme Johnson
Captain, USAF

AFIT/GE/ENG/89D-20

D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

89 12 14 040

AFIT/GE/ENG/89D-20

Payload Invariant Control via Neural Networks: Development and Experimental Evaluation

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Mark Alme Johnson, B.S.E.E.
Captain, USAF

December, 1989

Approved for public release; distribution unlimited

Preface

Exploring uncharted regions in science requires the support and assistance of many people. My deepest appreciation goes to Dr. Michael B. Leahy Jr., who first presented the problem and then provided encouragement and guidance throughout the research effort. He is also the driving force behind the robotics research currently being performed at the Air Force Institute of Technology. A special thanks go to Dr. Steven K. Rogers and Capt. Dennis Ruck for their assistance and insights into the theory and operation of neural networks. I would like to thank Dr. Gary Lamont for his guidance and support on hardware related issues.

Many issues in science remain unexplored due to a lack of the required backing. Therefore, I wish to express my gratitude to the Armstrong Aerospace Medical Research Laboratory Robotic Telepresence Program for looking to the future and sponsoring this research. Mr Dan Zambon, of the Information Systems Laboratory, deserves a special thanks for keeping the computers running and providing any possible assistance. I would also like to thank Mr. John Cholsa of the AFIT Model Fabrication Shop for producing the masses used in the experiments.

Contributions to research come in many forms. I wish to acknowledge the camraderie of Vernon Milholen and Samuel Sablan. We threw around many ideas and gave each other valuable support and advice. In closing, I wish to thank my parents for raising me to think as an individual and pay attention to the ideas of others.

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

Mark Alme Johnson

Table of Contents

on For
A&I
ced
ation
tion/
Availability Codes
Dist Avail and/or Special
A-1

List of Figures

List of Tables

Abstract

A new form of adaptive model-based control is proposed and experimentally evaluated. An Adaptive Model-Based Neural Network Controller (AMBNNC) uses multilayer perceptron artificial neural networks to estimate the payload during high speed manipulator motion. The payload estimate adapts the feedforward compensator to unmodeled system dynamics and payload variations. The neural nets are trained through repetitive training on trajectory tracking error data. The AMBNNC is experimentally evaluated on the third link of a PUMA-560 manipulator. Tracking performance is evaluated for a wide range of payload and trajectory conditions and compared to a non-adaptive model-based controller. The superior tracking accuracy of the AMBNNC demonstrates the potential of the proposed technique.

Payload Invariant Control via Neural Networks: Development and Experimental Evaluation

I. Introduction

1.1 Motivation

One problem in robot control is how to obtain accurate high speed trajectory tracking when the payload varies throughout the performance of the task. A solution to the problem is one requirement for realizing a manipulator capable of duplicating human performance. A manipulator with the ability to emulate human performance is one prerequisite for achieving Air Force Robotic Telepresence program objectives.

1.2 Objective

A research initiative of the Air Force Institute of Technology (AFIT) robotics research program is the development and evaluation of control methods which may lead to achieving human arm performance. Current research is centered on developing techniques that are robust and quickly adapt to payload variations [43,45]. An adaptive model-based control structure provides the framework for that research. The goal is to seek out a combination of an adaptive feedforward compensation approach and a robust feedback method which provides the best possible performance for a vertically articulated manipulator operating in an uncertain payload environment.

Adaptive feedforward compensation research is concentrated on looking at techniques to identify and estimate payload variations during task performance.

System dynamics are updated with the payload estimate to produce adaptive feed-forward compensation during manipulator motion. In a previous effort a stochastic multiple model adaptive estimation scheme was used to estimate the payload mass [76].

The payload mass estimation problem can be posed in pattern recognition terms. Neural networks can successfully solve various pattern recognition problems. If neural networks can quickly and accurately recognize robot tracking error patterns as a function of payload variation, they may provide performance superior to stochastic estimators.

The objective of my thesis research is twofold:

- to develop a mechanism employing neural networks capable of estimating a payload mass value from robot trajectory tracking error data, and
- to integrate the neural network payload mass estimating mechanism into an adaptive model-based control structure.

Subsequent experiments will explore the potential performance improvement using the resulting adaptive controller. All performance experiments will be performed on a PUMA-560 robot. The principal end product is the realization of a neural network technique for achieving autonomous adaptation of a robot manipulator to payload variations.

1.3 Problem Statement

The mechanism responsible for adapting to payload variations throughout task performance is the control system. Controlling a robotic manipulator is complicated by the coupled and nonlinear nature of robot dynamics. Contemporary industrial controllers model each joint as linear second order systems, and use speed and payload restrictions to bound dynamic coupling effects. The resulting critically damped systems are able to operate to about 60 percent of their full

electro-mechanical potential [42]. Including dynamic parameters in the controller design reduces speed and payload limitations [45,6,34].

Model-based control provides excellent trajectory tracking performance when accurate payload information is available [76]. However, payload variation information may be unavailable, unreliable, or unpredictable in intelligent robotic applications. Adaptive model-based control schemes attempt to provide accurate payload information via some adaptation technique. Two adaptation approaches used in current research to estimate payload parameters for adaptive robot control are based on the Lyapunov theory and Multiple Model Adaptive Estimation (MMAE) [72,70]. The MMAE adaptation technique uses a series of Kalman filters whose plant models span the payload parameter space. A bridge exists between Kalman filter techniques and the backpropagation training method used with multilayer perceptron neural networks [66]. The Lyapunov adaptation method is based on the premise of non-increasing energy in a mechanical system which guarantees control stability and implies the steady state errors go asymptotically to zero. Neural networks are globally Lyapunov adaptive systems [10].

Artificial neural networks (ANNs) are simplistic models of anatomical, physiological, behavioral, and cognitive aspects of animal biological processes [11,16]. Neural networks have shown potential for speech, vision, motor and motor-sensor control, tactile control, and other attributes required by robots to emulate humans. Previous use of neural networks to improve manipulator path following performance concentrated on replacing either the entire control system or the feedforward controller and/or prefilter with neural networks [10,21,25,30,56,38]. No previous network applications have considered payload variation. Many of these previous applications use the ability of neural networks to recognize patterns in data. The payload estimation problem can be posed in pattern recognition terms. This motivates the question: If the neural nets can quickly and accurately recognize the robot trajectory tracking position error patterns, as a function of payload

variation, can they provide performance superior to other estimation schemes?

Choosing a neural network involves answering several questions, "Can a network be found that can perform the required tasks, and is it available?" In addition, "If a network can not be found, can one be designed to perform the required tasks?" The answer to these questions is based primarily on the local availability, proven previous applications, and prior performance of the various neural network types. Once a particular network topology is chosen, the next task is developing a strategy to present trajectory tracking error data to the chosen neural network structure. The design should enable the neural nets to employ their proven classification abilities to determine the payload mass parameter from tracking error information patterns.

The model-based controller tracking performance with full payload information is known [43]; therefore, any unmodeled dynamic variations are assumed to be due to payload variations. Knowledge of the robot properties allow my research to focus on the neural network estimation scheme and explore the robustness of using neural networks to estimate the payload mass parameter. Therefore, once the neural networks prove able to detect the payload from trajectory error information, the search focuses on finding the ANN structure characteristics which yield the best training and potential operation performance. The criteria used to decide upon a specific neural network arrangement is which composition gives the highest accuracy and lowest error at the end of training. The next hurdle is to integrate the neural network payload estimation scheme into an adaptive model-based control structure.

The integration is dictated by the: chosen form of neural network, trajectory error data input method, and the manner of presenting the payload estimate to the control system. Once the synthesis is accomplished, experiments and analysis to ascertain the performance potential of the new technique must be performed. Experimental evaluation of known model-based controller performance versus adaptive

model-based control schemes is vital for realistic comparisons. Neural network performance as part of an adaptive model-based controller must be compared against the known model-based controller performance around which the adaptive controller is built.

1.4 Method of Approach

The multilayer perceptron (MLP) artificial neural network is chosen primarily due to local availability and existing resident knowledge of the net structure and operation. For the same reasons the backpropagation method is used to train the neural nets. There are no temporal based neural networks readily available and time constraints precluded attempting to implement one. The multilayer perceptron structure is commonly used for static classification problems. The application being considered is the dynamic movement of a PUMA-560 joint three.

The PUMA-560 is used as a case study because: the PUMA robot is available, the dynamics are well known, and it is a good case to study for Air Force applications. Competent experimental studies of adaptive robot control designs are scarce; however, in the following development all design and analysis work is based on experimental data. Therefore, only one joint of the PUMA is used as a starting point. A performance baseline trajectory for AFIT robot control research moves joint three through 105 degrees in 1.5 seconds. The trajectory is sampled every 4.5 milliseconds yielding 334 sample periods. Using only joint encoder position information yields one desired and one measured position input per manipulator joint for input to the neural networks at each sample period.

Two possible methods using multilayer perceptron neural networks are to use one net for the entire duration of the trajectory or use separate nets arrayed throughout the trajectory time period. Using one net is not feasible since the net would require 668 input nodes for a sampling rate of 4.5 milliseconds during a 1.5 second trajectory. How to obtain a temporal payload estimate from such a

structure is unknown. The other possible approach uses individual neural networks for each sample period throughout the trajectory. Using an array of individual nets is feasible because each neural net uses standard MLP feedforward operation. Also, the approach lends versatility to the final structure because the number of nets used and the rate of sampling the nets are easily varied.

Before going further in the design process, actual tracking position error data is analyzed to determine the viability of using only position error information with the neural networks. However, the error data distribution properties indicate the data portrays accepted pattern recognition information traits such as separation of characteristics in the position error parameter space. PUMA-560 runs are made with various payloads and controller payload information to make up feature vector training sets. A feature vector is a vector made up of characteristic elements (desired and actual joint positions) that identify a particular class of a parameter (payload) is a given parameter space. The payload position is known with respect to the PUMA-560 base reference frame. Payloads consist of three circular brass disks with 15.24 centimeter diameters representing one, two, three kilograms.

During these experiments payloads from zero to three kilograms are attached to the sixth link mounting flange of the PUMA. The payload is a function of ten variables. The variables represent the total mass, position of the mass with respect to the robot base reference frame, and inertia matrix terms [48, pages 407–419]. However, prior experimental evaluations have shown that the point mass assumption is valid for the first three links of a PUMA-560 manipulator. Therefore, the payload mass is assumed to be a point mass throughout the experiments and analysis [45].

Payload identification tests performed during training which yield accuracy and mean squared error measurements are used to track neural network training progress. The nets are considered trained when the accuracy and error outputs stabilize to constant values. The ability of the neural networks to train is validation

of the initial hypothesis that neural networks are able to detect the payload class from trajectory error data. As an initial development neural nets are trained for every ten trajectory sample periods.

To use the neural nets for on-line operation the nets are first loaded into their proper arrays during system initialization. Next, the position error information is presented to the nets to obtain a payload class decision. The third task is to be able to use the neural network payload decision to update the system dynamics information during on-line operation.

For each chore in the preceding paragraph, a simulation module or program is developed that enables troubleshooting and testing for proper function operation. Once all three functions are working individually they are incorporated into a simulated manipulator environment. They are tested in simulation to work out any problems not apparent during individual module testing. Following demonstrated successful loading and feedforward operation in simulation, the individual modules are integrated into the operational robot control environment.

After working out any remaining integration problems, the new adaptive model-based controller using neural networks as the adaptation mechanism is ready for experimental evaluations. Initial experiments concentrate on trajectory tracking and final position error performance of the adaptive versus the known non-adaptive model-based controllers. In addition, performance repeatability is tested by making ten manipulator runs with each payload variation.

For intelligent robotic applications the trajectory may not be known with certainty. The neural nets are trained on only a single trajectory. However, a modification is made that enables the new adaptive controller to be initial position and trajectory independent. Before being presented to the neural networks for either training or feedforward operation the position data is normalized around zero using means and standard deviations computed during training data formulation. The means and standard deviations are also loaded during system initialization.

That modification adjusts the means to the position of the manipulator prior to manipulator motion. The adjustment is experimentally shown to allow the manipulator to operate over trajectories with which the neural nets are not trained. The balance of the experiments focus on looking at trajectory tracking performance on a range of alternative trajectories. These experiments include: sample rate variations, payload range tests, and different methods of presenting the payload estimate to the system dynamics.

1.5 Thesis Contributions

Development and experimental evaluation of a new approach to adaptation to payload variation during high speed robot motion is the principle thesis achievement. An accomplishment central to achieving the adaptation ability is the development and experimental verification of a technique using artificial neural networks to detect and identify payload mass parameters from manipulator trajectory position error data. A Neural Network Payload Estimator (NNPE) and Adaptive Model-Based Neural Network Controller (AMBNNC) are the prototypes developed as vehicles to validate the proposed techniques.

Software and hardware tools required to support current and future research were design and either programmed or made. The software tools enable the quick transition of neural net weights from the training environment to on-line operation. The hardware tools that were designed and obtained are the masses and various attachments used to attach the masses at different locations and distances with respect to joint three.

Analysis of experimental results indicate neural networks are able to determine payload mass parameters from trajectory tracking error information. As part of an adaptive model-based control structure, neural networks are able to quickly and accurately determine the payload mass parameter during high speed manipulator motion over a range of trajectories, sample rates, and payloads. These findings

demonstrate the potential of the NNPE to provide the quick payload adaptation required for human arm emulation.

1.6 Organization

The balance of the thesis consists of four chapters and three appendixes. Chapter 2 sketches the information reviewed in a manner that takes one through the progression of thought used to discern the concept for the thesis. Chapter 3 reveals the development of the Neural Network Payload Estimator and the Adaptive Model-Based Neural Network Controller. The experimental evaluation of both mechanisms is the subject of Chapter 4. Chapter 5 summarizes the experimental results so as to shed light on their meanings to the thesis goals and future research. The appendixes complement or extend the thesis substance.

Appendix A completes a comprehensive review, began in Chapter 2, of how neural networks are being applied in robotics research. Appendix B proposes a temporal implementation of the multilayer perceptron structure and includes a modified backpropagation training algorithm along with usage discussion. Appendix C presents some of the troubleshooting done to resolve problems encountered during the research. All code and additional experimental data is located in Technical Report ARSL-89-12 [32].

II. Background Information

2.1 Introduction

Development and evaluation of artificial neural networks and robot control techniques are areas of active research. Studies into using neural networks to solve robot control problems are beginning to emerge. To appreciate a fresh approach to a robot control problem requires looking at: the problem, strategies currently used to address the problem, and the new method. The problem being addressed is how to estimate an unknown and/or varying payload mass during high speed robot motion to provide payload invariant gross motion trajectory tracking. Current methods directed at the problem use various parameter estimation techniques to determine the payload mass from one or more sensor outputs. The new technique uses neural networks to estimate the payload mass parameter using trajectory position errors derived from joint encoder measurements. To present a foundation from which to understand the new method, the following discussion focuses on reviewing artificial neural networks, robot control, parameter estimation, pattern recognition, and current applications of artificial neural networks to robot control.

2.2 Neural Networks

The application of existing neural network models to various computational problems is an active area of research. The attractiveness of neural networks stems from their many inherent characteristics, including fault tolerance, the ability to process many hypotheses at the same time, and their ability to learn from and adapt to changing situations [49]. Artificial neural networks (ANN) model anatomical, physiological, behavioral, and cognitive aspects of animal biological processes [11,16]. The models relate respectively to the topology, node characteristics, training, and learning aspects of the various neural models [49]. Hecht-Nielson estimates there are over fifty different types of neural networks [26]. Current efforts

in designing artificial neural networks are based on modeling observed behavior of biological neural systems. Hecht-Nielson states there are 47 identified distinct traits of human behavior and current ANN architectures use at most five or six of them [26].

Some ANN architectures are based on the observed operation of single neurons. Figure 2.1 shows one example of a biological neuron and Figure 2.2 depicts its electrical circuit analog (an artificial neuron). On the simplest level operation of the two systems is similar. Both receive inputs from many other processing elements and yield a single output based on some function of the inputs. Some current artificial neural network models consist of layers of artificial neurons operating in parallel. Figure 2.3 shows a simple Neural Network structure. The bottom layer consists of the network input nodes. The next two layers are called *hidden layers*. The top layer contains the output nodes. An existence proof by Kolmogorov states that a three layer network (two hidden layers) with $N(2N+1)$ nodes can compute any continuous function of N variables [49, page 18]. Therefore, most applications use two hidden layers.

Algorithms and circuits are designed to perform the functions specified by models. Development of neural network models are based on how biological neural systems appear to work. The ability of an algorithm or circuit to fully duplicate the performance of the modeled system depends on the model accuracy. Model accuracy is directly related to the understanding of the system being modeled. Presently, a rudimentary understanding of neural system behavior is emerging. Patricia Churchland, in an article on perspectives on cognitive neuroscience, aptly states the situation:

> Even if we could simulate, synapse for synapse, our entire nervous system, that accomplishment, by itself, would not be the same as understanding how it works. [13].

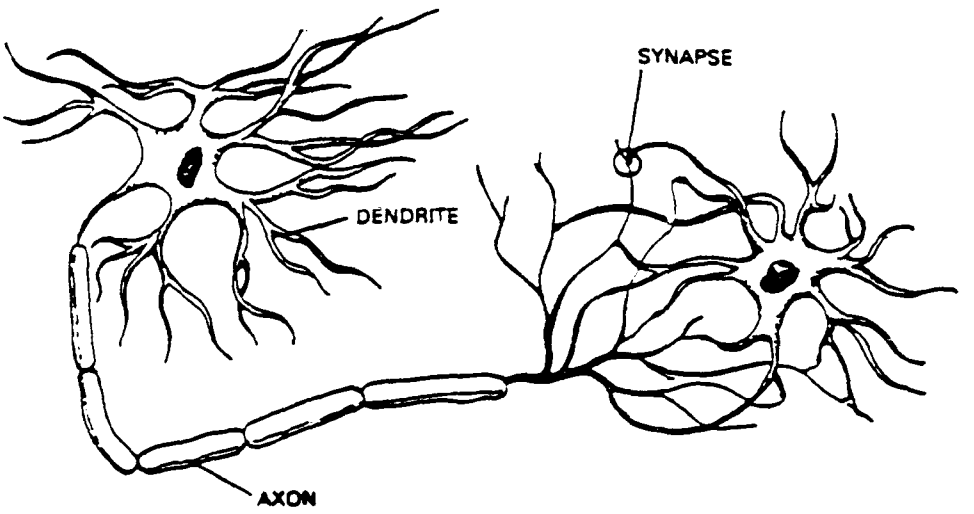

Figure 2.1 A Biological Neuron [14]

To better understand the operation of an artificial neural network the next two sections cover two processes modeled on one current understanding of biological neural behavior. Presented first is the operation of the feedforward algorithm of a multilayer perceptron (MLP) neural network. Feedforward operation is followed by a look at the Back-Propagation training algorithm. Both developments can be found in many references [49,65,74,4,67,38]. The equations covered are out of *An Introduction to Computing with Neural Nets* by Richard P. Lippmann [49].

2.2.1 Neural Network Operation The general feedforward operation of an artificial neural network is illustrated in Figure 2.4. At each node in the figure, the weighted inputs to a nodes are summed with the node threshold and then passed through a sigmoidal nonlinearity. The node output is used as either an input to each node in the next layer of nodes or as an output of the neural net. In some applications the input and output processes at each node are quite complex and consist of complicated mathematical representations. For the example shown in

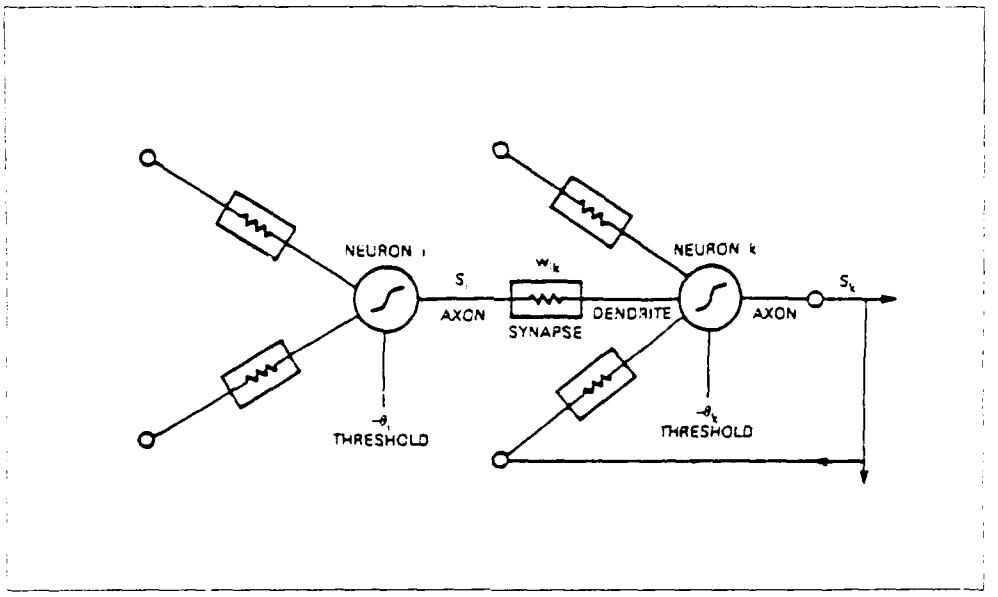

Figure 2.2. An Artificial Neuron [14]

Figure 2.4, the node operation is governed by:

$$y = f\left(\sum_{i=0}^{n-1} w_i x_i + \theta\right) \tag{2.1}$$

where:

- y is the output of the node,
- n is the number of inputs to the node,
- x_i is the ith node input,
- w_i is the ith input weight, and
- θ is the node threshold.

Each node's output function $f(\bullet)$ is the sigmoidal function in Equation 2.2.

$$f(\bullet) = \frac{1}{1 + e^{-(\bullet)}} \tag{2.2}$$

During operation, feature vectors are presented to the neural network as inputs. The outputs are the net decision as to the class represented by the input. For the neural net to indicate the correct class, the network must be trained with a representative sample from the classes from which the net will make decisions.

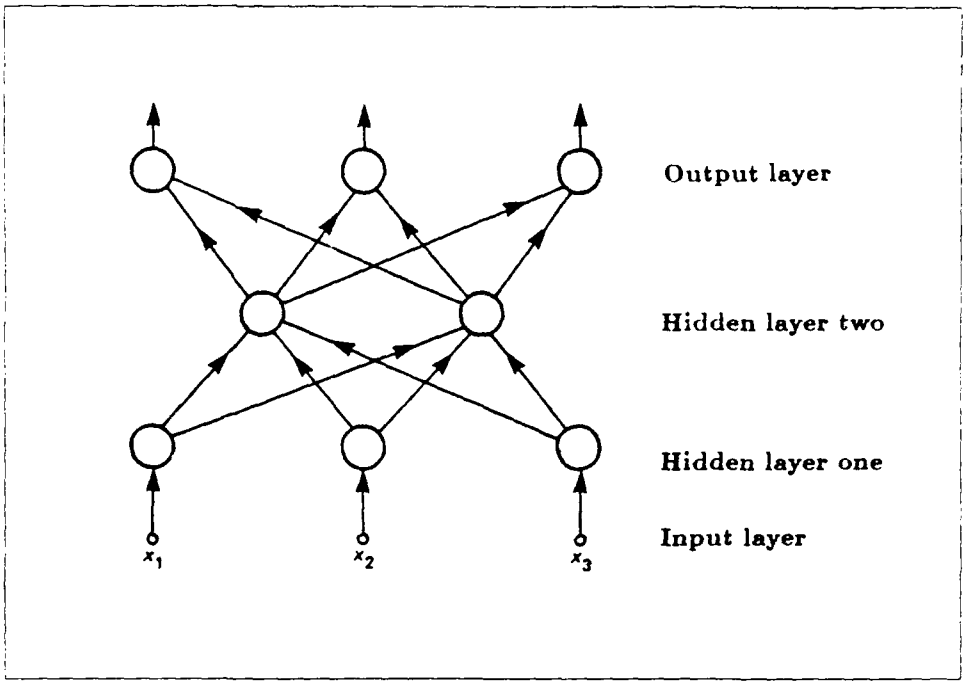

Figure 2.3. A Simple Artificial Neural Network Structure [75]

2.2.2 Neural Network Training One common algorithm for training multilayer perceptrons is Back-Propagation, developed in 1974 by Paul J. Werbos for his Harvard PhD thesis, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* The method is currently recognized as either the Generalized Delta Rule or Back-Propagation [49].

To train a multilayer perceptron using the Back-Propagation algorithm the weights are initially set to random values. Next, vectors consisting of a feature vector and the desired output are presented to the network. The output of each node is governed by the sigmoidal:

$$f(\eta) = \frac{1}{1 + e^{-(\eta)}} \tag{2.3}$$

where the activation function, η, is chosen to facilitate searching the feature space [64]. The value of η is application dependent [3]. The difference between the actual and desired network output is an error signal used to adjust the weights.

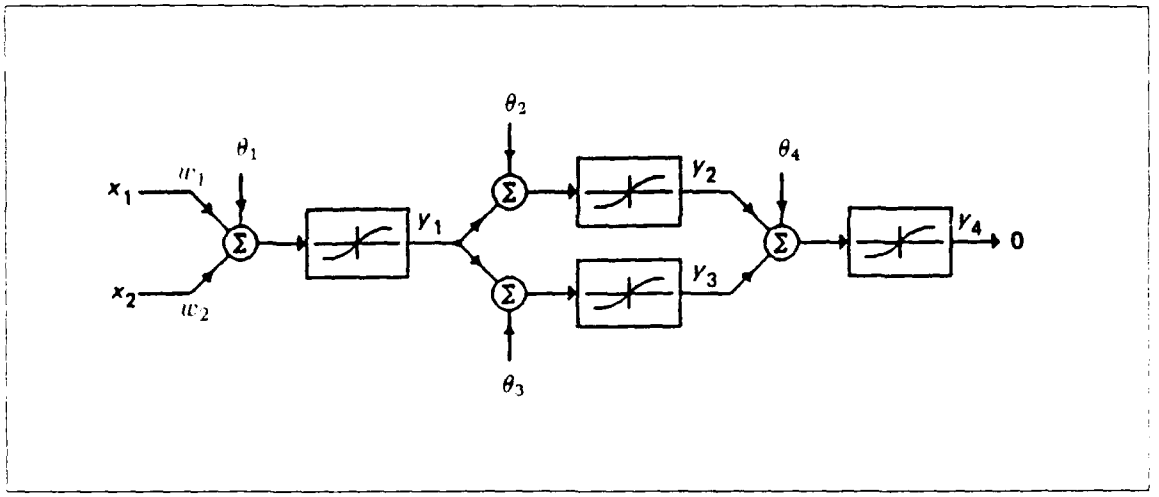

Figure 2.4. Multilayer Perceptron Feedforward Operation Flow Diagram

Weights are changed using the equation

$$w_{i,j}(t+1) = w_{i,j}(t) + \check{x}_i \delta_j \chi + \alpha(w_{i,j}(t) - w_{i,j}(t-1)) \tag{2.4}$$

where:

- $w_{i,j}$ is the weight between the lower layer's ith node and the next layer's jth node,
- χ is the training rate,
- $\check{x}_i$ is the output of the lower layer's ith node,
- δ_j is the error term for node j, and
- α is the training momentum factor,

The training momentum factor, a number between zero and one, is used to prevent being trapped in a ravine crossing the weight space by determining the direction to take with the next training step [67]. The training rate χ is a number between zero and one used to indicate the step size to use during training [67].

For all nodes except the output nodes δ_j is computed by

$$\delta_j = \check{x}_j(1 - \check{x}_j) \sum_k (\delta_k w_{i,k}) \tag{2.5}$$

where $\check{x}_j$ is the node j output, and the summation is over all the previous layer nodes. For the output nodes δ_j is determined by:

$$\delta_j = y_j(1 - y_j)(d_j - y_j) \tag{2.6}$$

Here y_j is the actual and d_j is the desired node j output. As each training vector is applied to the network, each connection weight is recursively updated from the output layer towards the input layer using the criteria presented above. Each node threshold is adjusted in a similar manner. When the weights and thresholds stabilize for input feature vectors representing all the classes, the network is considered trained and every weight and threshold is fixed. The previously outlined feedforward operation is used with the trained network for classifying unknown feature vector inputs.

2.2.3 Why Neural Networks? Neural networks are successful in pattern behavior recognition applications such as speech and vision processing [79]. Many of the parameters required for compliant motion control and control in uncertain environments can be determined from current sensor information using pattern behavior recognition techniques. Parameter knowledge improves performance of robot control systems [41,76,62,57]. Application of artificial neural network pattern recognition techniques to intelligent robot control problems may yield realistic solutions to complex problems. The potential performance improvements of using pattern recognition techniques via neural networks for robot control are unknown.

2.3 Robot Control

2.3.1 Introduction Over 40 years ago the scientific world was shocked to rediscover that feedback control systems exist in nature. The feedback loop was "identified in society, men ,animals, and machines" [58]. Today, the control system is at the core of intelligent robotic systems research and development. Apart from

a robot's physical design constraints, the control system determines the manipulability and tasks the robot is able to perform.

2.3.2 Background Information The physical design of an industrial robot is determined by the task(s) it will perform. Robotic manipulators are typically made up of mechanical links put together with rotary and/or prismatic joints. The number of degrees of freedom (DOF) is one measure of a robot's manipulability. Three positional DOF define the manipulator workspace, and three orientation DOF determine the ability to align the end effector (gripper, tool, etc.). Redundant links or joints are added to increase strength or range; however, they may not add to manipulator mobility. Mobility is a measure of how many of the positional and orientation degrees of freedom are actually available in a given manipulator structure. Therefore, the ability of the manipulator to position and orientate the end effector is directly related to the number and structure of the manipulator's mechanical links and joints. Serial and/or parallel mechanical link configurations are used depending on the planned tasks.

A serial link manipulator is shown in Figure 2.5. It's defining criteria is that each actuator is either at or associated with one DOF or joint [71]. Serial link designs are typically used in applications where a relatively light tool is used, such as spray painting automobiles. Parallel link designs are more applicable to heavy lifting jobs such as lifting and placing loaded pallets. A parallel actuated arm can be identified by two or more serial chains connecting the base either partially or fully with the final link [71]. A manipulator with a parallel linkage is presented in Figure 2.6.

Robot drive systems use electric, pneumatic, and/or hydraulic actuators. The actuators deliver the drive to the joints directly or use a gear train and/or belt system. To perform a given task a device called an end effector is attached to the last link. If the task involves grasping and holding objects it is called a gripper.

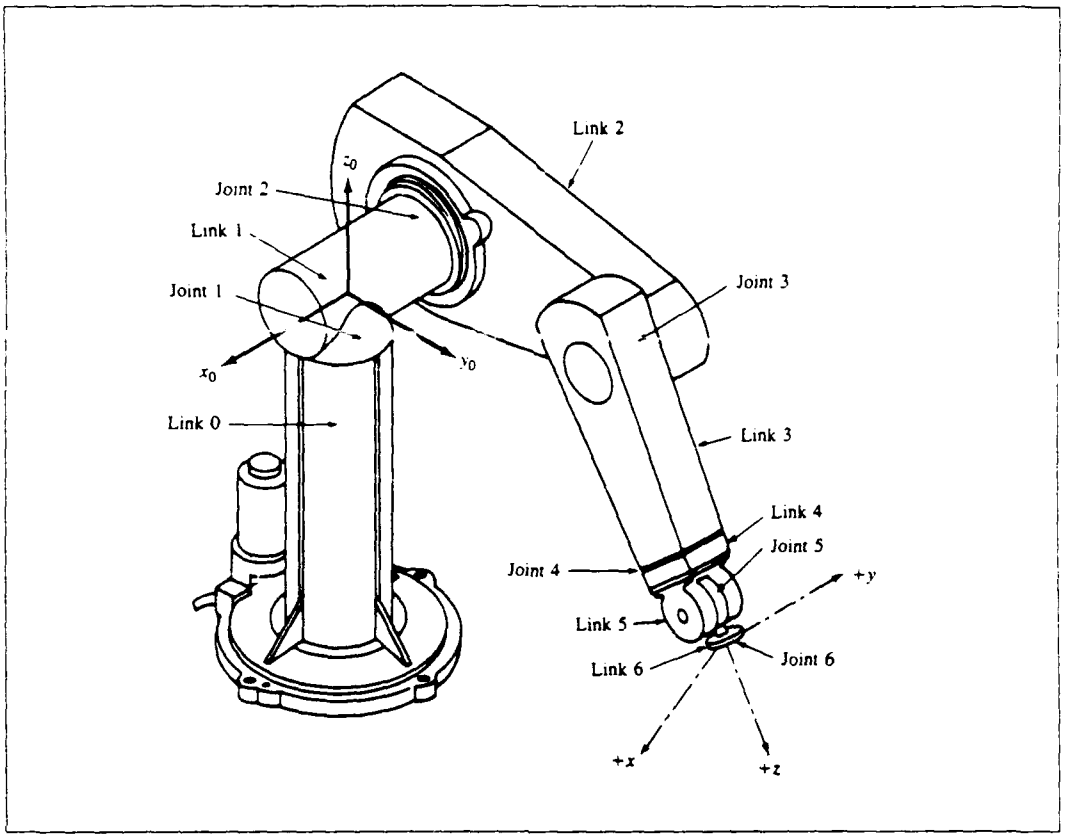

Figure 2.5. A Serial Link Manipulator: PUMA [20, page28]

An end effector designed to perform work is called a tool [42].

The mathematical equations describing the motion of an individual link are established in the mechanics of rigid bodies. For a single link these equations of motion are able to be represented by a linear differential equation. However, when the equations of motion are expanded to include several connected links, the differential equations used to describe the motion become complex, nonlinear, and coupled. These equations of motion become the basis for the design of manipulator control systems. Conventional approaches to formulating the dynamic equations of motion are based on Lagrange-Euler and Newton-Euler equations of motion [20, page 82]. The Lagrange-Euler equation is [42]:

$$\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} \qquad i = 1, 2,...,n \tag{2.7}$$

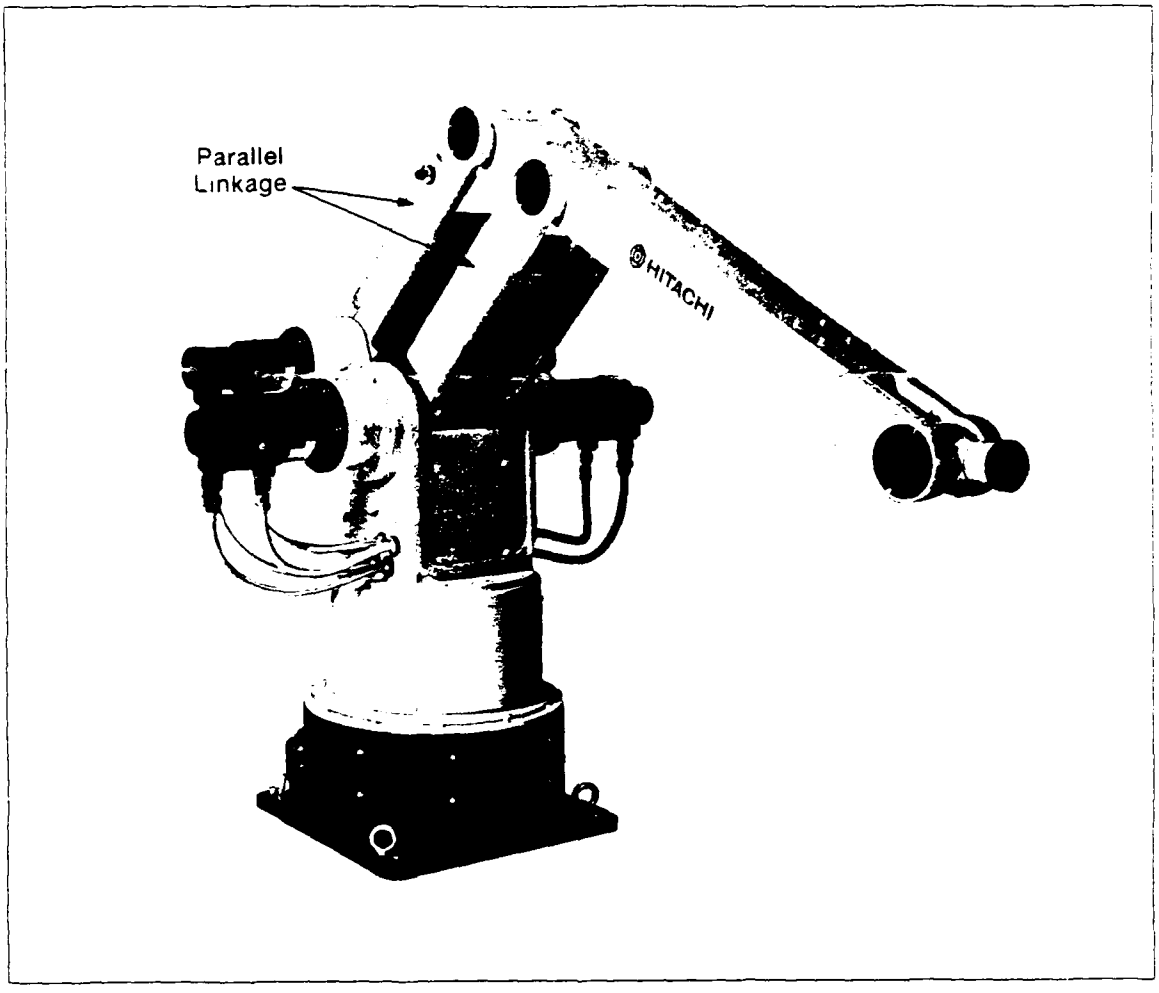

Figure 2.6. A Parallel Link Manipulator [60, page 39]

where:

- L =total robot arm kinetic energy K - total robot arm potential energy P,
- q_i =generalized robot arm coordinates,
- $\dot{q}_i$ =first derivative in time of the generalized robot arm coordinate, q,
- τ_i =generalized torque/force applied to joint i.

The most common generalized robot coordinate system is the Denavit-Hartenberg coordinate representation. The Denavit-Hartenberg system uses matrixies to define the translational and rotational relationships between adjacent links [20]. For an excellent presentation of the Denavit-Hartenberg coordinate system and how to

establish one for a given manipulator see *ROBOTICS: Control, Sensing, Vision, and Intelligence* by K. S. Fu, R. C. Gonzalez, and C. S. G. Lee [20].

Using the Lagrange-Euler formulation and including terms for an external payload the equations of motion for a manipulator become [76]:

$$N\Upsilon(t) = [D(q,a) + N^2M]\ddot{q} + h(\dot{q},q,a) + N^2B_m\dot{q} + \tau_s + g(q,a) \qquad (2.8)$$

where:

- n =the number of links of the manipulator;
- $q, \dot{q}, \ddot{q}$ = n-vectors of joint angles, velocities, and accelerations, respectively;
- a = m-vector of the unknown load parameters;
- $N = n \times n$ diagonal matrix of gear ratios for each joint($\frac{motor\ velocity}{link\ velocity}$);
- $D(q,a) = n \times n$ matrix of manipulator load and position dependent inertias;
- M = diagonal $n \times n$ matrix of actuator inertia terms;
- $h(\dot{q},q,a) = n$-vector of centrifugal and coriolis torques;
- $\tau_s = n$-vector of static friction torques;
- $B_m = n \times n$ diagonal matrix of damping coefficients;
- $g(q,a) = n$-vector of gravity loading terms; and
- $\Upsilon(t) = n$-vector or joint motor torques.

2.3.3 Current Directions in Robot Control Research The role of the robot control system is to maintain a desired response throughout the performance of some function. Robot control is complicated by the extensively coupled nonlinear nature of robot dynamics. Contemporary industrial robot controller designs attempt to bypass manipulator dynamic parameters by considering each link or joint as a linear second order system. Dynamic coupling is treated as a disturbance and is bounded by using speed and payload restrictions. The result is a critically damped system capable of achieving perhaps 60 percent of it's electro-mechanical potential [42].

Many of the efforts to improve the capability of robotic systems focus on incorporating knowledge of the manipulator dynamics into the controller design. A controller designed with this knowledge is able to achieve greater speeds and carry heavier payloads. Inclusion of manipulator dynamic parameters into controllers is possible due to the advances in computational power. Knowledge of the parameters enables the controller to compensate for system nonlinearities and coupling while performing a given task.

Current research is involved with developing techniques to control manipulator motion when knowledge of manipulator parameters is unknown or uncertain. Many schemes have been proposed that tackle the problem from differing perspectives. Fu, Gonzoles, and Lee categorize them as joint motion controls, resolved motion controls, and adaptive controls [20, page 202]. The balance of the section will highlight a blend of the joint motion controls and adaptive controls techniques known as adaptive model-based control. Adaptive model-based control is chosen due to the experimentally proven ability of model-based control methods to improve tracking accuracy over high speed trajectories [5,46]. Furthermore, combining proven performance with the facility to adapt to the task and environment is crucial when addressing intelligent robotic applications.

2.3.4 Adaptive Model-Based Control Model-based control schemes attempt to compensate for dynamic nonlinearities and modeling discrepancies by using feedforward dynamic compensation and feedback techniques. Feedback techniques seek to eliminate disturbances caused by modeling inaccuracies [41,33]. Feedforward dynamic compensation tries to eliminate perturbations caused by known dynamic interations of gravity, coupling torques, friction, centrifugal and coriolis forces, and payload information. Compensation is achieved by producing nominal feedforward torques which locally linearize the plant so that a classical feedback technique can asymptotically drive the error to zero. Figure 2.7 diagrams a typical model-based control system.

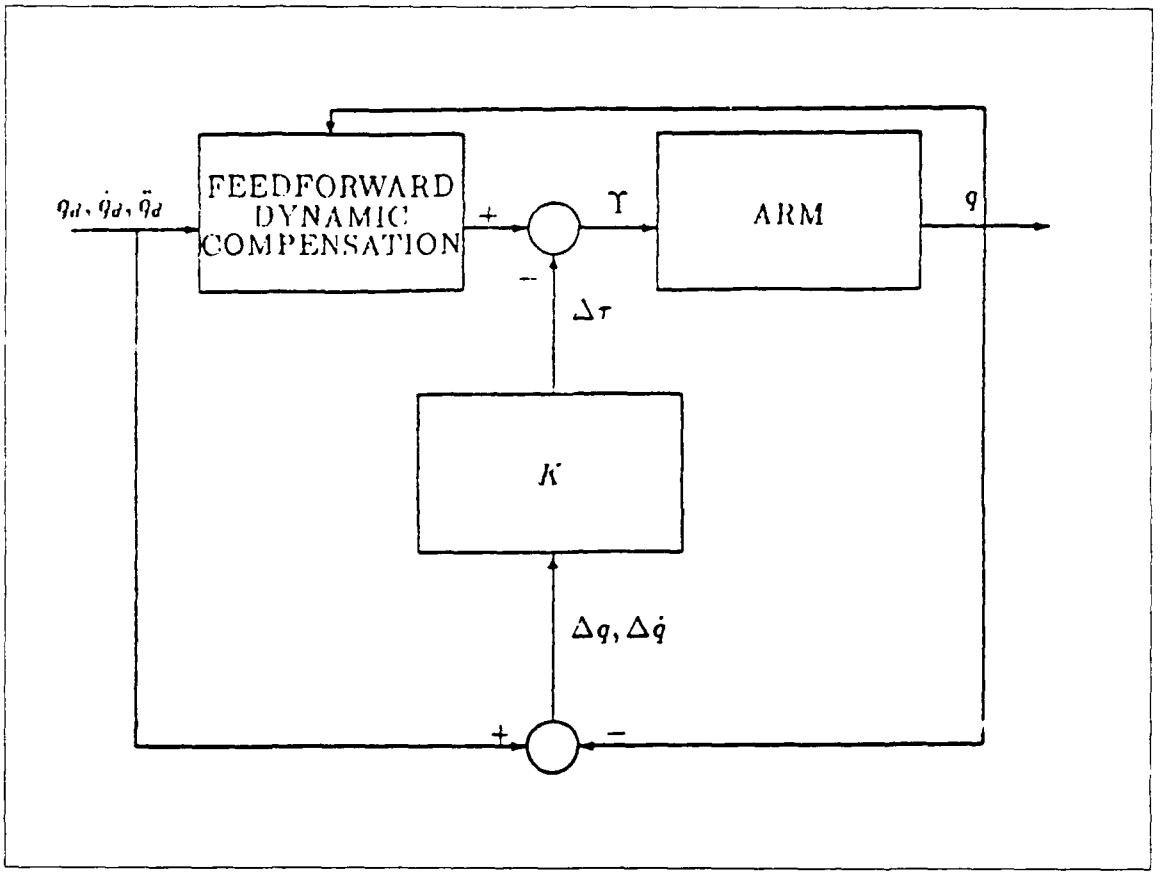

Figure 2.7. Model-Based Control System Block Diagram

Model-based control systems provide excellent trajectory tracking performance when accurate payload information is available [41,76]. However, lack of accurate payload information causes serious degradation of tracking accuracy [43]. For intelligent applications, payload information may be sparse or nonexistent. Also, the control system must be able to adapt to variations in payload information throughout the performance of any task. These are reasons for the development of adaptive model-based control schemes.

Adaptive model-based control schemes seek to provide accurate payload information via some adaptation technique. The adaptation techniques rely on payload parameter estimation schemes which use position, velocity, acceleration, or other types of available sensory information. Payload parameter estimation

techniques are usually based on least-squares, stochastic, or Lyapunov methods [47,72,52]. Assuming knowledge of all parameters of each manipulator link, the task is to estimate the parameters of an external payload and incorporate the estimate into the manipulator dynamic equations of motion. The estimated parameters are included in the equations of motion as indicated in Equation 2.8 by the variable $a(t)$. Ways of estimating parameters for an adaptive model-based controller, illustrated in Figure 2.8, are covered next.

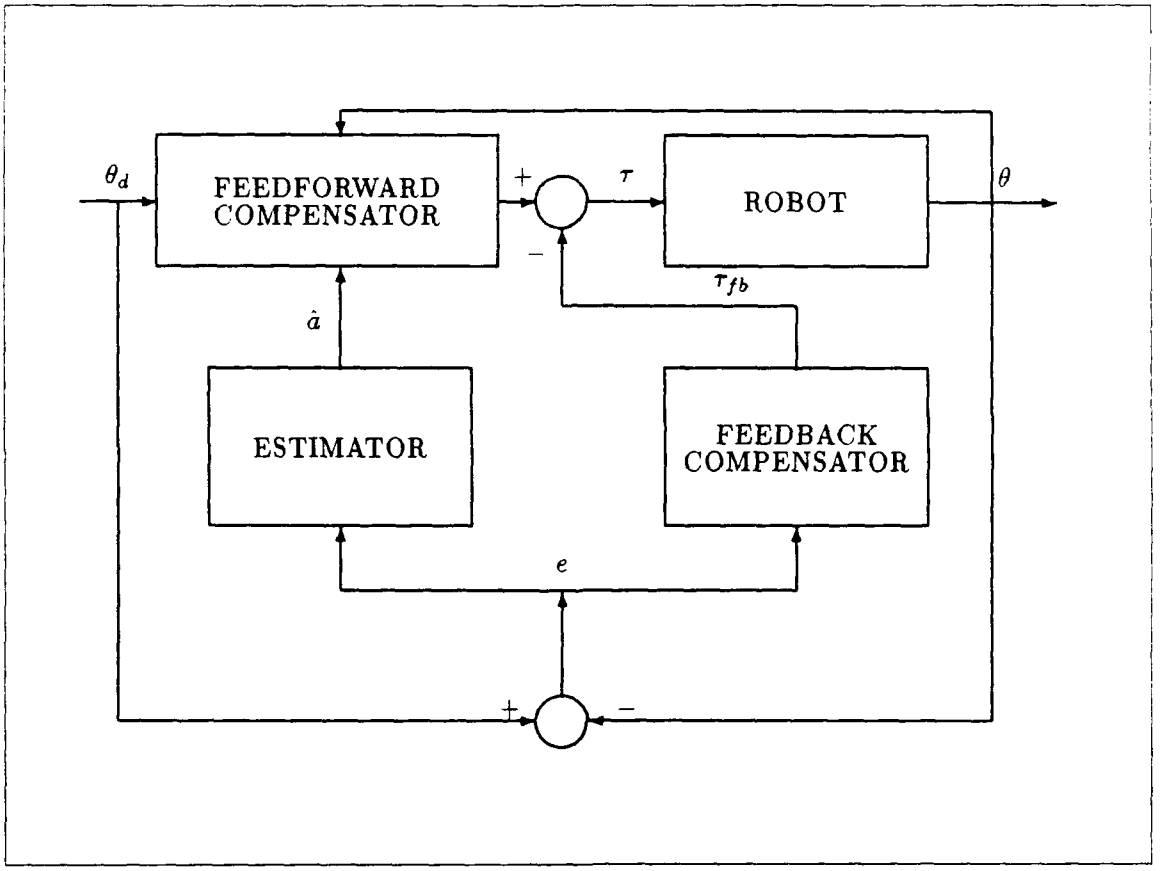

Figure 2.8. Adaptive Model-Based Controller Block Diagram

2.3.5 Estimating Parameters The objective of all robot parameter estimation methods is to enable an adaptive controller to achieve global convergence for all desired initial conditions and trajectories. Most methods of estimating robot parameters rely on a form of least squares computation [78,9,7,6,55,27]. An ex-

cellent review of the least squares method of estimation is found in reference [47]. However, standard least squares techniques have many pitfalls when used in on-line applications.

Standard least squares techniques suffer from asymptotic convergence and the inability to adapt to quickly changing situations [47]. The inability to adapt is due to their gain going quickly to zero. Also, standard least squares techniques require linearization of the parameters about some nominal values which may be either guessed at or chosen from experience. However, experience may not suffice when faced with a novel situation. An area of continuing research is improving estimation techniques to overcome some of the least squares technique fallacies.

Some researchers use a gain resetting method in an effort to improve the least squares facility to handle system perturbations [22,23]. The gain resetting scheme's shortfall is that all the past experience contained in the controller gain matrix is lost when the gains are reset. An improvement is to use a forgetting factor [27] that causes the past controller gains to die out at a predetermined rate. However, the forgetting factor is usually found by trial and error or 'informed' guessing. Li and Slotine improved the forgetting factor method by implementing a gain-adjusted-estimator (GAF) that yields exponential convergence in simulation [47]. The GAF estimator uses the norm of the gain matrix, which does not go to zero, to adjust the evaluation of the estimator error. The GAF estimator is a promising linear estimator.

In control systems engineering the Kalman Filter is considered the premier linear estimator.

> It combines all available measurement data, plus prior knowledge of the system and measuring devices, to produce an estimate of the desired variables in such a manner that the error is minimized statistically [53, volume 1, page 5].

The advantage of using Kalman filters is due to their ability to account for measurement and system noises. The primary disadvantage for on-line use is the computational loading, especially for the extended Kalman filter. In addition, the choosing of the filter gains is still considered by many to be a 'black art' [51]. One method of avoiding the need for an extended Kalman filter is to use an array of linear Kalman filters that span the given parameter space. The technique is known as the Multiple Model Adaptive Estimator (MMAE).

Application of MMAE techniques to robot control problems is an ongoing research effort at AFIT. Larry Tellman successfully combined a MMAE with a model-based controller to form an adaptive model-based perturbation controller [76]. In operation, manipulator trajectory position error characteristics are matched to a MMAE Kalman filter tuned to detect the error properties associated with a particular payload variation. The payload estimate is used to update the system dynamics information during robot motion.

The above approaches yield a specific value for the payload parameter. Other adaptive control methods use an indirect indication of the payload parameter. One technique based on Lyapunov theory is the Model Reference Adaptive Controller (MRAC) which compares the actual position, velocity, and/or acceleration with a reference model. The payload estimate is embedded in the differences between the actual and reference model outputs which are used to adjust feedback loop gains. The closed-loop control system is guaranteed to be asymptotically stable due to the Lyapunov basis of the design. However, the MRAC is unstable in high speed motion applications [37].

Limitations and possible improvements of the above techniques are explored in several areas of current literature. One recent article by Gordon Kraft compared a least squares control application, a Lyapunov based MRAC, and a neural network based controller. Simulation results in the article indicate that each method is good at performing some functions and poor in performing others [37]. Another

investigator, Dennis Ruck, recently forged a bridge between Kalman filter techniques and the backpropagation training method [66]. The use of neural networks for control applications is burgeoning. Current studies focus on the ability of neural networks to learn and recognize patterns in control structures and sensory information. Therefore, before launching into how neural networks are being studied for robot manipulator control, a review of pattern recognition is presented.

2.4 *Pattern Recognition*

2.4.1 Introduction Intelligent robots operating in autonomous applications will be required to make decisions based on incomplete information and uncertainty. Furthermore, the decisions must satisfy task constraints. In addition, there are many tasks an autonomous robot must be able to do individually or all at once. A representative sample of these tasks include: directional guidance, obstacle avoidance, orientation determination, range finding, object identification, path planning, and sensorimotor coordination [77,11]. Pattern recognition or classification will play a key role in the decisions made about given tasks.

Pattern recognition is identifying an object according to measurements of the object's distinguishing features [29]. Several methods of performing pattern classification exist in classical decision theory. The section covers the terminology, basic approach, and an example of classical decision theory as applied to pattern analysis.

2.4.2 Background The terminology used within the pattern classification field takes its cues from vision analysis terms. First, each measurement is a feature. Next, a set of n measurements becomes a point in an n-dimensional feature space. Finally, a feature vector is formed by connecting the associated features. Determining what features of an object to select for decision making is one key to obtaining a meaningful decision.

The basic approach to pattern recognition starts by choosing discriminating characteristics for each entity to be classified. Then the chosen characteristics are measured and used to define class memberships. From statistics of each characteristic feature, decision thresholds are established. These thresholds form regions within the multidimensional feature space. One assumption is that feature vectors of different class members form separate clusters in the feature space. Unknown entities are classified by where their feature vectors map to in the feature space.

2.4.3 Classical Decision Theory In classical decision theory, the crux of all decision making schemes is how to partition the feature space to provide for accurate object classification. One simple method is known as the nearest-neighbor classification. The method searches the entire feature space and classifies the unknown entity by the nearest example (neighbor) it finds to the object. Mathematically, if in a feature space for some feature F of some i and j,

$$|F_{i,j} - X| < |F_{m,n} - X| \qquad \text{for all } m \text{ and } n, \tag{2.9}$$

then the unknown X is ascribed to class i [29, page 337]. The advantage of this method is that clusters can have complicated shapes in the feature space. However, it is computationally intensive and cluster overlapping can reduce decision accuracy. An example of a simple feature space is shown in Figure 2.9.

Several schemes are available that attempt to address the computational and overlap problems. One method, known as nearest centroid classification, reduces the computational load by assuming the examples of each class form round non-overlapping clusters. The centroid of the class the unknown feature vector is nearest to then becomes it's classification. The method is often extended to the use of probability density models for clusters of differing shapes to define a class. When using these methods it is useful to choose geometrical shapes that are mathematically tractable and adequately define the clusters. Neural networks

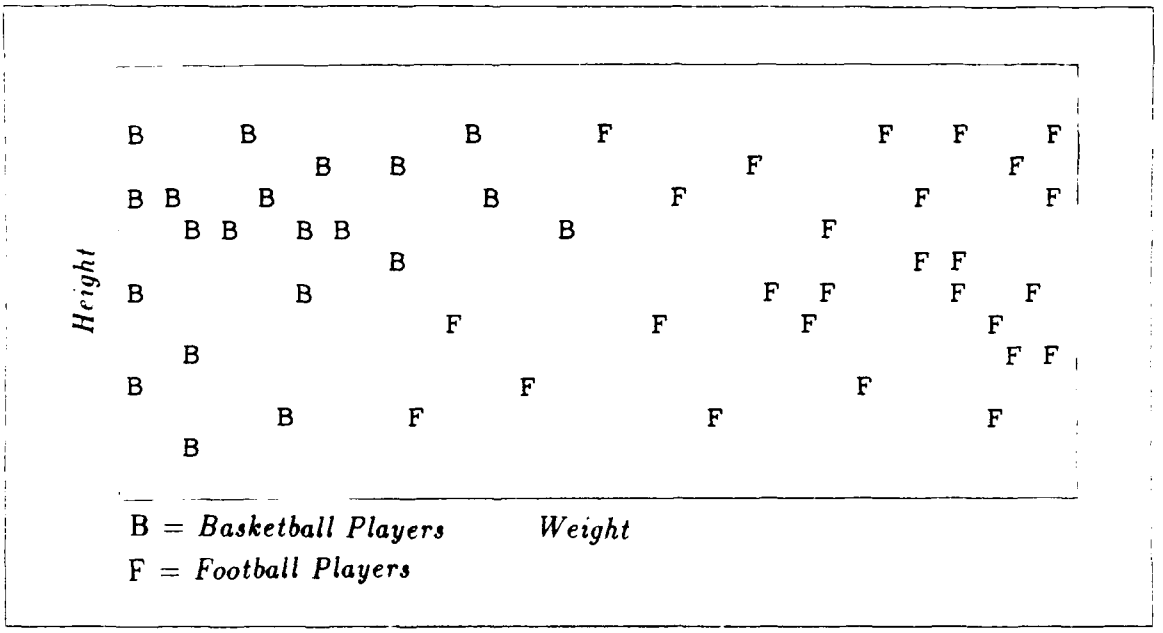

Figure 2.9. A Simple Feature Space

are non-classical in that they form their own decision regions in the feature space during their training.

2.5 Manipulator Dynamics Based Trajectory Control with Neural Nets

In manipulator trajectory control the idea is to use the neural network attributes to improve the tracking of robot manipulators, particularly in circumstances of uncertainty in the task performance. Two principal approaches are currently taken to apply neural networks to robot controllers. One approach is to use the ANN as a high level controller in place of the entire controller. The other method uses the ANN as a feedforward controller and/or prefilter, after teaching it the inverse dynamics. The first example takes an information processing point of view to develop a high level controller.

Bill Horne and M. Jamshidi use data representations to a Cerebellar Model Articulation Controller (CMAC) to control a 1 degree-of-freedom gripper attached to a Rhino XR educational robot [30]. They use three types of data representations:

position (between the gripper fingers), control movement (using duty cycle and count parameters), and the relationship between position and control. Network nodes are assigned to each position in the work space. The network is then trained with a reinforcement method to move from one position to another using step inputs. After training, the step inputs are used to move the gripper fingers. When a move is commanded for which the system is not trained, a binary search tree of preset relationships is used to perform the commanded position change. Figure 2.10 shows a representative portion of the binary search tree. Experimental results are ambiguous. The authors conclude by suggesting refinements, such as improving the position/control relationship.

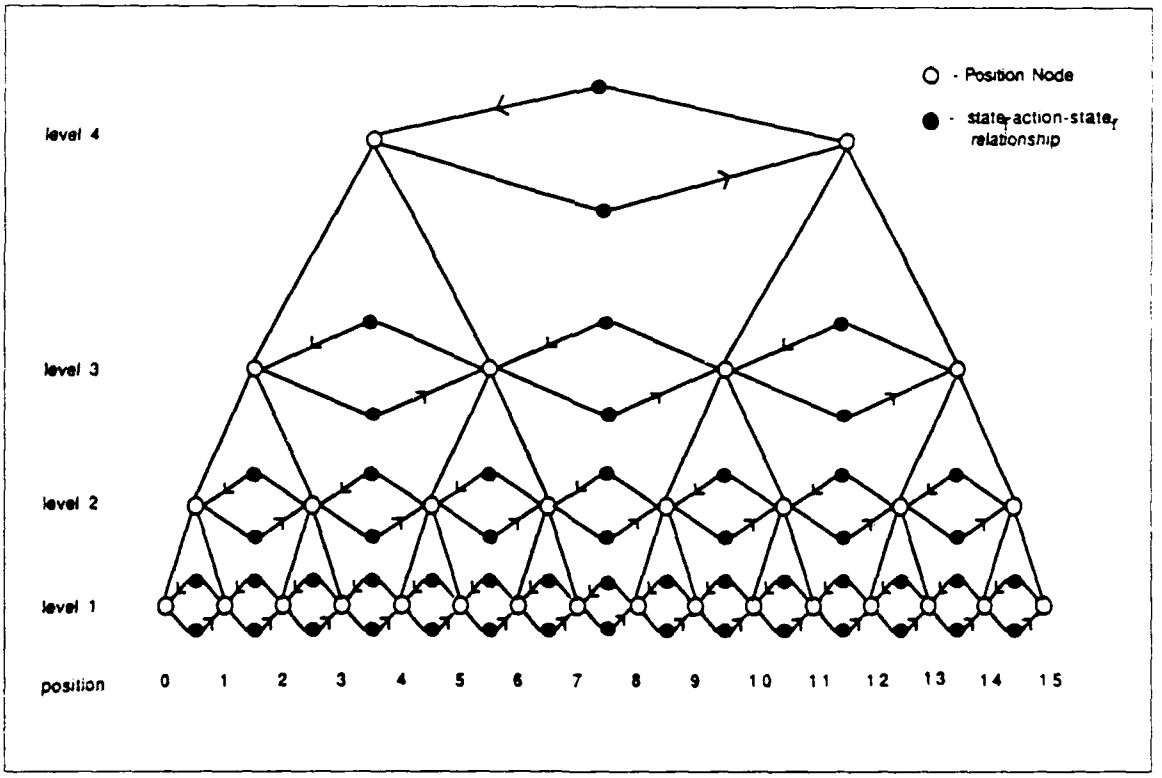

Figure 2.10. Binary Search Tree [30]

The majority of on going efforts in robot trajectory control focus on training the neural network the inverse dynamics of the system over a representative range of its operation and then using it as a feedforward controller/prefilter. Representative

of these efforts are Goldberg and Pearlmutter's use of a multilayer perceptron to learn the dynamics of the CMU Direct Drive Arm II [21], Guez and Ahmad's use of a multilayer perceptron to solve the inverse kinematics problem for a two link planar manipulator [25], and Atkeson and Reinkensmeyer's use of an Associative Content Addressable Memories (ACAM) scheme as the feedforward controller with the MIT Connection Machine in simulating a planar two-joint robot and a running machine [10].

A representative but novel method is currently under investigation by W. Thomas Miller et al, at the University of New Hampshire [56]. They are essentially piggybacking a CMAC network with a modified weight adjustment law onto a constant gain industrial robot controller to adaptively enhance the trajectory following capabilities of a General Electric P-5 industrial robot. The system dynamics involved are simplified by the parallel linkage and relatively low joint velocities of the P-5 robot.

The CMAC functions as a feedforward controller in parallel with the fixed plant. Slow speeds are used on moderately benign trajectories to evaluate the system performance. The authors mention using this scheme for nonrepetitive trajectories yet only test on repetitive trajectories. Hopefully, further system evaluation using nonrepetitive tasks will be forthcoming. The control scheme may be useful on repetitive tasks in environments subject to small perturbations.

In an experimental application, Akin and Sanner of MIT applied neural networks as a prefilter as part of a *Neuromorphic Pitch attitude Regulator of an Underwater Telerobot* [2]. The major problem encountered was the computational loading when using a single microprocessor for the entire control scheme.

Extending the employment of neural networks to other parts of the control system structure, F. Pourboghrat and M. R. Sayeh propose using two neural networks, one as a feedforward controller and another as an adaptive state feedback controller. They state that "a feedforward controller, without any error feedback,

is not able to compensate for unpredicted disturbances" [61]. Initially presented as a learning controller, they suggest that the feedback controller will adaptively compensate for perturbations. Experimental verification was not accomplished. Figure 2.11 gives a pictorial view of their proposed structure.

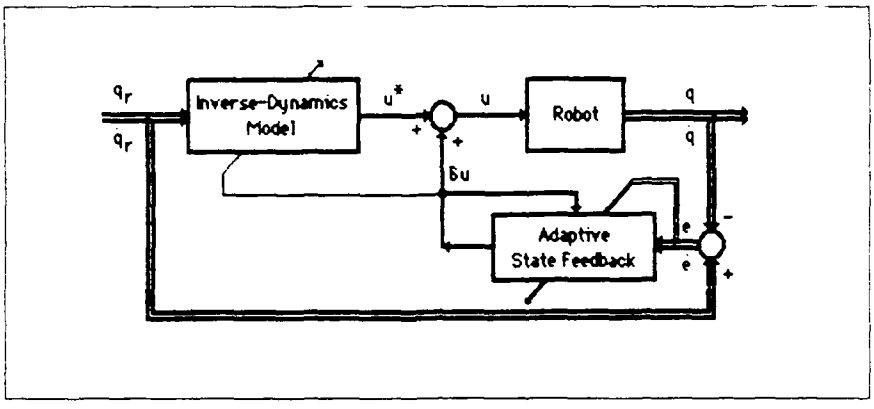

Figure 2.11. Hybrid Control Structure [61]

2.6 Summary

All of the above investigations are important to determine the potential tracking accuracy improvements from using neural networks for robot control. Control problems that are able to be posed in pattern recognition terms may find a viable solution using neural nets. One such problem is the payload estimation problem. Presenting trajectory tracking error profiles to neural networks for use in identifying payload mass parameter behavior is possible. Payload information obtained using multiple model adaptive estimation schemes via Kalman filters improves manipulator performance. Using payload information supplied by neural networks in place of other estimation schemes should produce a similar improvement in manipulator performance. Additionally, neural networks may be able to perform the payload estimation function more efficiently than stochastic or least squares methods and do the estimation over a greater range of payload variations. To realize an adaptive robot controller using neural networks as the adaptation mechanism, the

next chapter presents the development of a new method of using neural networks to provide a robot controller with payload information during task execution.

III. The Approach Taken

As in many engineering solutions, a compromise is the final result.

3.1 A Beginning

Central to achieving a neural network payload estimator is determining if an Artificial Neural Network (ANN) can identify a payload mass from trajectory error information. The initial task is determining the type of neural network to use. The available position information and how that data might be presented to a neural network are only two factors to consider in deciding on a type of ANN. Once a specific neural network type is chosen, the following is a list of issues needing resolution.

- Can the selected ANN correctly identify the payload masses?
- What neural network size is required for a given range of payloads and position data inputs?
- How is the correct size and/or structure determined?
- What is the best way to present the position data to the nets?
- Is neural network training time an important consideration?

Once these issues are resolved the selected neural network must be fully trained and tested.

For on-line use of neural networks in a robot controller, how to integrate the neural network operations with the controller functions is the first consideration. Related tasks include: loading the neural net weights during initialization, providing for their processes during on-line operation, and incorporating the neural net payload estimate into the feedforward dynamics compensator during robot motion. After achieving an on-line operational system the question of 'How neural networks

affect system performance?' must be answered. The following discussions describe the developments and methods used to realize an operational adaptive controller using artificial neural networks as the payload estimation mechanism.

Presented first is a technique that uses neural networks to provide a payload estimate from trajectory tracking error patterns. Next, issues on how to present the payload information to a robot control system are discussed. The chapter culminates in an example adaptive controller realization by illustrating the development of a direct adaptive model-based controller using neural networks as the adaptation mechanism.

3.2 Neural Network Payload Estimation

Developing a Neural Network Payload Estimator (NNPE) requires looking at the available sensor data containing trajectory tracking performance information. Ignoring vision and tactile information reduces available information to joint position encoder data. Trajectory tracking error is the calculated difference between the desired and actual position. However, is position error data suitable for use with neural networks; are there patterns in the data neural networks can learn? Experimental results given in the Chapter 4 indicate definite patterns exist in manipulator trajectory error data. Therefore, desired and actual position data is used to present deviations from the desired trajectory to the neural networks. Other possible inputs, such as velocity and acceleration, are not addressed in the development.

The multilayer perceptron (MLP) artificial neural network using backpropagation as a training method is used in the NNPE development. The multilayer perceptron structure is chosen primarily due to local knowledge and availability of the neural network. Multilayer perceptrons are inherently static systems. Previous applications of neural networks in robotics have been to static problems such as a fixed background for visual systems, or fixed models and trajectories for inverse

kinematics applications. To apply a static structure to a temporal problem requires adapting the structure in such a way that the abilities afforded by the mechanism are used in a dynamic environment.

The study of the problem starts with looking at a high speed 1.5 second trajectory which is sampled every 4.5 milliseconds and yields 334 sample periods during the trajectory. Each sample period requires one desired and one measured position input per manipulator joint. Several different ways of using the position information with neural networks to identify the payload are possible. One method is to use a single net trained over a given set of trajectories. To cover 334 sample periods, one net would require a sum of 668 input nodes per joint, an unknown number of hidden layer nodes, and however many required output nodes (depends on how the parameter space is divided). Producing a usable output from the net may not be possible due to the massive net size and temporal computation problems within the net.

Another method of using the position data is to use nets at each sample period of a set of trajectories. The nets would be trained with update period position data and temporal information throughout the trajectories (see Appendix 2 for further development of the idea). As an initial step towards realizing a temporal MLP, one neural network is trained for each update period of a trajectory and sampled at that time during manipulator motion. To cover 334 sample periods requires using 334 individual nets with two input nodes per joint, a much smaller number of hidden layer nodes, and the necessary output nodes.

Using the method of backpropagation training covered in Chapter 2 requires a representative set of data with which to train the neural nets. The data set must contain the actual and desired position information as well as the payload class associated with the data. Additionally, positional variations occur whenever a robot or machine is brought on-line or calibrated. Therefore, ten runs of the manipulator are made for each trajectory and each known payload variation to

generate training data. Five runs are made with a calibration between each run. The other five runs are made with one initial calibration to include the effects of performing repetitive tasks. Included in the training data collection are sets of runs where the controller is told a range of payload values other than the actual payload. Giving the controller an incorrect payload value produces tracking error information to use in training the nets to detect the actual payload from deviations from the desired trajectory.

Neural networks are usually presented with linearly normalized inputs between zero and one or negative and positive one. However, the information content of one trajectory position data vector is small. Therefore, during formation of training data sets the mean and standard deviation of all actual and desired position data points at the individual time frame are computed. Normalization around zero is achieved by subtracting the computed means from each position value and then dividing by the standard deviation. Payload class information is augmented to each training feature vector using 0.9 to indicate the desired neural net class output with all other class outputs set to 0.1. These values are used for indicating the class to the neural nets during training and are due to the use of the sigmoidal nonlinearity (see Equation 2.3) in the operation at each node.

Training data sets covering representative sample times of a given trajectory are used to find the neural net structure that works best with the trajectory position information. The number of output nodes is set by the number of increments used to divide the payload mass range. The number of input nodes is manipulator dependent. Two nodes per manipulator joint are used to input the desired and actual position data to the neural networks. After determining the number of inputs and outputs to use, the design problem becomes determining how many nodes to use in the hidden layers of a multilayer perceptron ANN.

Training accuracy and mean squared error results are used to track the training of the neural nets. Training results are generated at predetermined intervals

during training by testing the classification ability of the nets using the training data as a test set. The criteria used to choose the net composition for use in the payload estimation scheme is which design attains the highest training accuracy with the least error over the majority of the trajectory training data sets. Neural net 'fully trained' status is decided by either reaching a given limit on the number of training iterations or achieving a set level of accuracy and error. In many cases the neural nets stabilize around local or global minimums before reaching set training limits.

The ability of the neural networks to train proves their potential to discern payload information from trajectory tracking error information during on-line operation. Another measure of neural network payload classification potential is testing them using feature vectors not used during training. Accuracy and error are calculated in the same manner as during training to allow direct comparison of the results.

3.2.1 Presentation of Estimated Payload to a Controller Several schemes of presenting the estimated payload to the control system exist. One method is to update the payload value every time a new payload estimate is obtained. Updating the payload value with every estimate gives better tracking performance during the initial transient portion of a trajectory but may create problems later in the trajectory. The problems may occur because the neural networks are estimating the payload variations. Thus, even with a payload, if the arm is correctly tracking the desired trajectory the neural nets will indicate a payload of zero kilograms. The incorrect payload information may prevent the control system from adapting to an unanticipated disturbance.

Another strategy is to update the payload value only when a change in payload is detected. Updating the payload only when a change from a current value occurs is attractive in that once an initial payload value is established, it is un-

changed unless the payload characteristics change. Computational loading is decreased and performance is increased. One drawback is that the controller will never be told if the payload does go to zero kilograms.

Both of the above methods indicate a high confidence in the mass estimate. If the confidence is low, then a sliding window method with a majority voting scheme may provide adequate performance. However, there is an inherent lag in obtaining the payload estimate that may be prohibitive for high speed applications. The method where updates are performed only when detected payload values change is the technique used during development and initial performance evaluations. During later performance evaluations a compromise of the first two approaches is used.

3.3 Adaptive Controller Realization

The Neural Network Payload Estimator's (NNPE) sole purpose is to provide usable payload information to assist a controller performing required tasks in uncertain environments. The payload information is used to update the model of the system dynamics used by the dynamics compensator to compute the feedforward portion of the commanded torque sent to the manipulator. Figure 3.1 shows the placement of the NNPE in the structure of a model-based controller (MBC). The control mechanism is called the Adaptive Model-Based Neural Network Controller (AMBNNC).

The AMBNNC uses feedforward dynamic compensation and a Proportional-plus-Derivative (PD) feedback loop. Using a PD feedback loop is used to facilitate comparison of experimental results with previous AFIT research [76,68]. Including terms for payload parameters, the feedforward dynamic compensation (τ_{ff}) and PD feedback loop (τ_{fb}) are described by

$$N\tau_{ff} = [\hat{D}(q,a) + N^2 M]\ddot{q} + \hat{h}(\dot{q},q,a) + N^2 B_m \dot{q} + \tau_s + \hat{g}(q,a) \qquad (3.1)$$

$$\tau_{fb} = K_v \dot{e} + K_p e \qquad (3.2)$$

where:

- $q, \dot{q}, \ddot{q}$ = vectors of joint angles, velocities, and accelerations, respectively;
- a = vector of the unknown load parameters;
- $\hat{D}(q, a)$ = matrix of estimated manipulator load and position dependent inertias;
- N = diagonal matrix of gear ratios for each joint ($\frac{motor\ velocity}{link\ velocity}$);
- M = diagonal matrix of actuator inertia terms reflected through the gear train;
- $\hat{h}(\dot{q}, q, a)$ = vector of estimated centrifugal and coriolis torques;
- B_m = diagonal matrix of damping coefficients;
- τ_s = vector of static friction torques;
- $\hat{g}(q, a)$ = vector of estimated gravity loading terms;
- K_v = vector of velocity gains;
- K_p = vector of position gains;
- e = vector of position errors ($q_{desired} - q$), and;
- $\dot{e} = (\frac{e}{\Delta t})$, a vector of velocity errors.

The NNPE provides an estimate $\hat{a}$ of the payload parameter vector in Equation 3.1. The payload estimate is used to adapt the feedforward compensator to payload variations. Since the payload estimation is driven by trajectory tracking error AMBNNC is a form of direct adaptive control.

Figure 3.2 is an AMBNNC operational flow diagram. During system initialization the basic functions of providing power throughout the system, running system operation checks, and calibrating the system are performed. During task initialization the neural network weights, means, and standard deviations for each time frame used during the trajectory are loaded into their respective arrays. The means and standard deviations are the same ones computed during training set formulation. Figure 3.3 shows the structure of the neural net arrays for intervals spanning a given period of operation. As part of the task initialization the means are adjusted for differing initial positions using the following two step process:

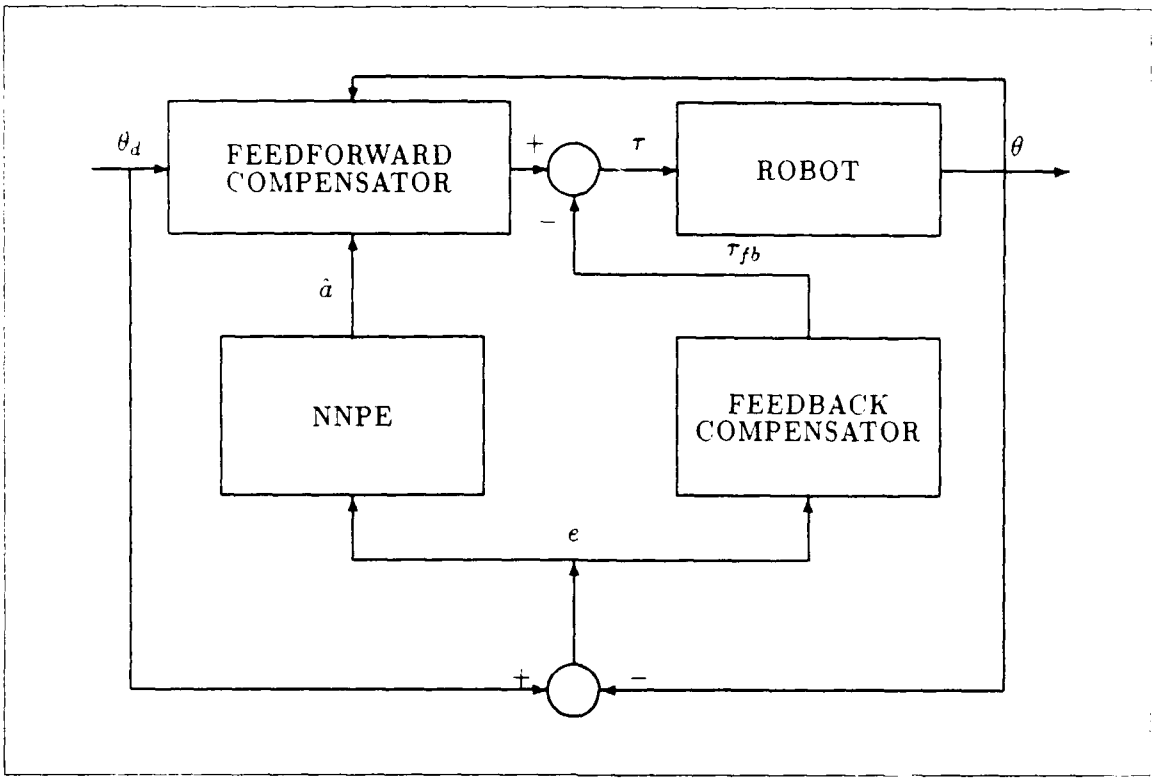

Figure 3.1. Adaptive Model-Based Neural Network Controller

- the calculated initial position is subtracted from the initial position of the trajectory that the nets were trained on, then
- the difference is subtracted from each desired and actual mean calculated during formulation of the training data sets.

Using the above process enables neural nets trained using one trajectory and initial position to be used for trajectories independent of the initial position. This is experimentally validated in Chapter 4. Also, during task initialization, instructions for task performance are stated, a value for the payload mass is given, and system dynamics are computed.

Nominal dynamics information is used to initialize the feedforward dynamic compensation torques. During manipulator motion the sample time, and desired and actual position information are presented to the ANN feedforward algorithm.

The sample time is used as a pointer to the set of weights to use in the neural network feedforward operation. The position information is normalized and input to the neural network. The decision of the net is indicated by the highest output node which is translated into the associated payload class. The payload class is used to either recompute or leave unchanged the mass parameters using a point mass assumption to modify all payload parameters.

A compromise between two approaches is used to change the payload mass parameters. For the early part of the trajectory, the mass parameters are recomputed every time the nets are used. For the latter two-thirds of the trajectory, the mass parameters are only recomputed when there is a detected change other than to zero. The compromise is due to the performance gained from each method in differing payload and trajectory situations. Combining the two techniques forms an adaptation mechanism within the controller that attempts to drive the trajectory tracking error to zero irregardless of the payload. Adaptation is produced by on-line recomputation of system dynamics.

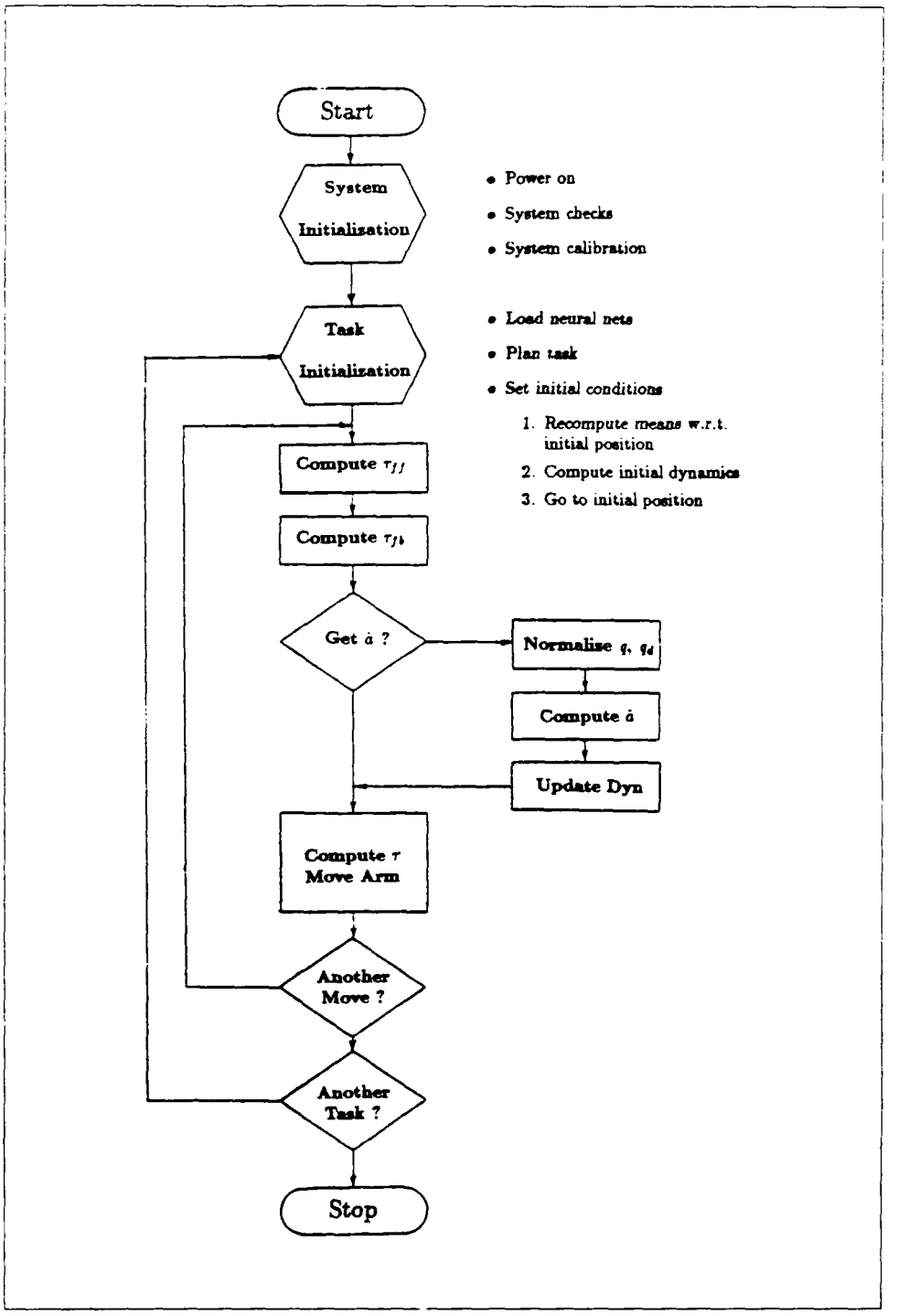

Figure 3.2. AMBNNC Operation Flow Diagram

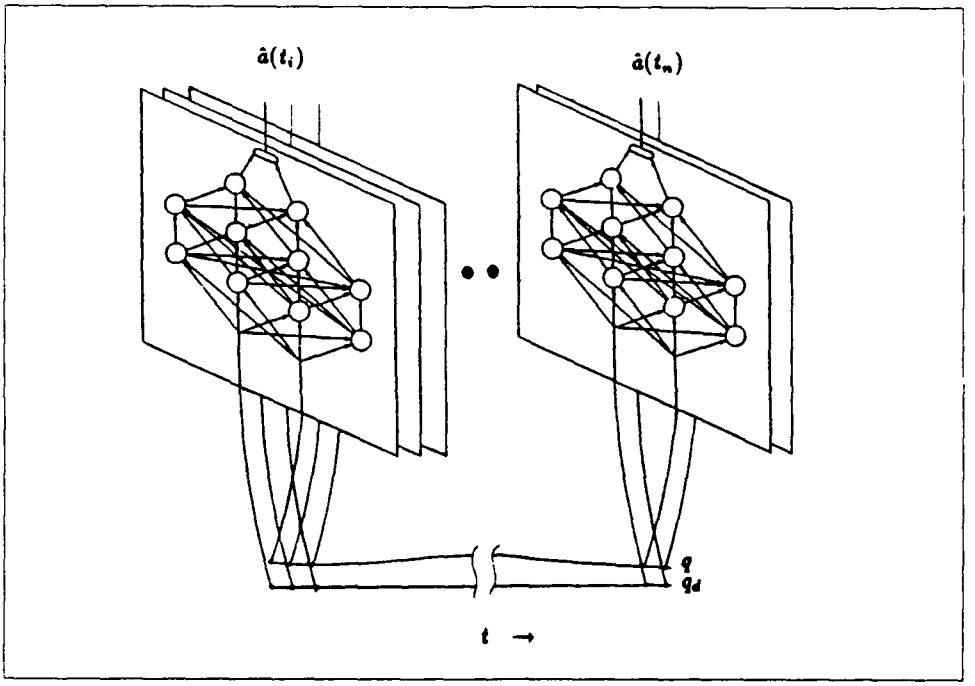

Figure 3.3. Temporal Arrays of Neural Networks

3.4 Summary

The first known development of a technique to estimate payload mass parameters from robot trajectory position error data using artificial neural networks has been presented. Some ways of presenting the payload estimate to a robot control system during on-line operation were covered. Finally, the formulation of an adaptive model-based controller (AMBNNC) using neural networks as the adaptation mechanism (NNPE) was realized. An experimental evaluation of the AMBNNC performed on a PUMA-560 manipulator along with other tests used to make development decisions are presented in the following chapter.

IV. Experimental Analysis

4.1 Introduction

The goal of the following dialogue is to demonstrate and validate the ability of an artificial neural network (ANN) to provide accurate estimates of payload mass during high speed manipulator motion in uncertain environments. The true test of the validity of a new development is experimental evaluation in a known environment. Accordingly, a profile of the experimental environment begins the discussion followed by the tests that examine the ability of neural networks to discern payload mass from trajectory error data. The subsequent section sketches the forging of the Adaptive Model-Based Neural Network Controller (AMBNNC). The balance of the chapter surveys the performance of the AMBNNC versus a known single (non-adaptive) model-based controller (SMBC).

4.2 Experimental Environment

A PUMA-560 operating under the ARCADE environment [43] is the test platform used to generate the neural network training data and perform subsequent AMBNNC evaluations. ARCADE is resident on a VAXstation III (ROBBIE) and uses both parallel and serial interfaces with the PUMA LSI-11/73 computer. A DRV11-J parallel interface is used to pass angular position information and motor current information between the VAXstation and the LSI-11/73. The LSI-11/73 is used only as a preprocessor. Device driver software is provided by VAXlab software which is layered on top of the host VMS operating system [43].

The nominal dynamics of the PUMA-560 are well known along with the dependence of tracking accuracy on payload information [45,43]. The single (non-adaptive) model-based controller (SMBC) evaluated in previous studies [43] is used to generate the training data and provides a known baseline against which to

compare AMBNNC performance. The AMBNNC uses the same feedforward and feedback algorithms as the SMBC except the Neural Network Payload Estimator (NNPE) supplies the payload estimate required for adaptation. The Proportional-plus-Derivative (PD) feedback loop gains (see Equation 3.2) used during testing are tabulated in Table 4.2. Due to communication limitations, a maximum servo rate of 4.5 milliseconds (222 Hz) is used throughout the evaluations except where tests are performed to examine the affects of using a range of sample rates.

Neural network training is accomplished on a MicroVAX III using ADA software written by Dennis Ruck [65]. The ADA software implements the backpropagation training algorithm given in Equations 2.3 - 2.6. Prior to training each net is seeded with numbers from a random number generator, and the training rate, η, and momentum, α, are set to 0.3 and 0.7, respectively. Upon reaching the area of a stable minimum, α is reduced to 0.5 or 0.4 to enable finer searching and convergence.

The ARCADE environment is modified to include algorithms for: loading the neural net weights, means, and standard deviations; adjusting the means to the computed initial position prior to arm motion; and performing the feedforward operations given in Equations 2.1 and 2.2. The algorithms are written in FORTRAN and hosted on ROBBIE within the ensemble called *Neuroboto*.

Link i	Position (K_{p_i})	Velocity (K_{v_i})
1	640.0	72.0
2	1331.0	129.0
3	360.0	25.0

Table 4.1. PD Feedback Gains

4.3 NNPE Development and Validation

Initial tests examine the dispersion of the PUMA link three trajectory error data to determine the suitability of using the data with neural networks. Position error is the calculated difference between the desired and actual position as given in Equation 4.1.

$$e(t) = q_d(t) - q(t) \tag{4.1}$$

Position error and neural network training data are produced by moving the third link of the PUMA through -105 degrees in 1.5 seconds from an initial position of (-50, -90, 210) degrees. A minimum jerk trajectory generator is used to calculate the trajectory [43]. Figure 4.2 shows the trajectory position, velocity, and acceleration profiles. Payloads range from zero to three kilograms in increments of one kilogram. To generate a representative set of training data, the manipulator is run through the trajectory ten times for each payload condition indicated in Table 4.2. Each payload is a 15 centimeter brass disk attached to the sixth link mounting flange (shown in Figure 4.3) with the difference in mass being a function of disk thickness.

Figure 4.1 contains plots showing the position error scattering of the tracking error data for eight sample periods of the trajectory. In each plot there are ten error terms for each payload mass increment from zero to three. For example, in each plot a small circle represents the occurrence of an error value during a manipulator run when the difference between the actual payload and what the controller is informed to be the payload is one kilogram. Each occurrence of an individual error term is counted 'up' the vertical scale in each plot such that a small circle on the vertical axis 2 indicates the second time that error value occurred. Plots *B* through *E* show that the payload class/tracking error data is separated in the feature space. Overlapping of the error data for the payload classes is seen at the very beginning (plot *A*) and towards the end of the trajectory (plots *F* through *H*). The dispersion patterns within the feature space indicate that the data is suitable

to use with neural networks. Error data dispersion and overlapping are reflected in subsequent neural network training and operation accuracy and error results.

Payload	*Initial Controller Payload (in Kg)*					
(in Kg)	0	1	2	3	4	5
0	*	*	*	*	-	-
1	*	*	*	*	*	*
2	*	*	*	*	*	*
3	*	*	*	*	-	-
* indicates variations tested.						

Table 4.2. Payload Conditions.

Trajectory		*Initial Position* (link three)	*Distance Moved* (in degrees)	*Time Taken* (in seconds)
1	A	210.0 deg.	- 105.0	1.5
	B	180.0 deg.	- 105.0	1.5
	C	135.0 deg.	- 105.0	1.5
	D	90.0 deg.	- 105.0	1.5
2	A - D	-	- 105.0	1.7
3	A - D	-	- 105.0	1.9
4	-	0.0 deg.	+ 105.0	1.5
5	A - D	-	- 52.5	1.5
Links 1 and 2 are at -50.0 and -90.0 degrees, respectively.				

Table 4.3. Trajectories used for Testing

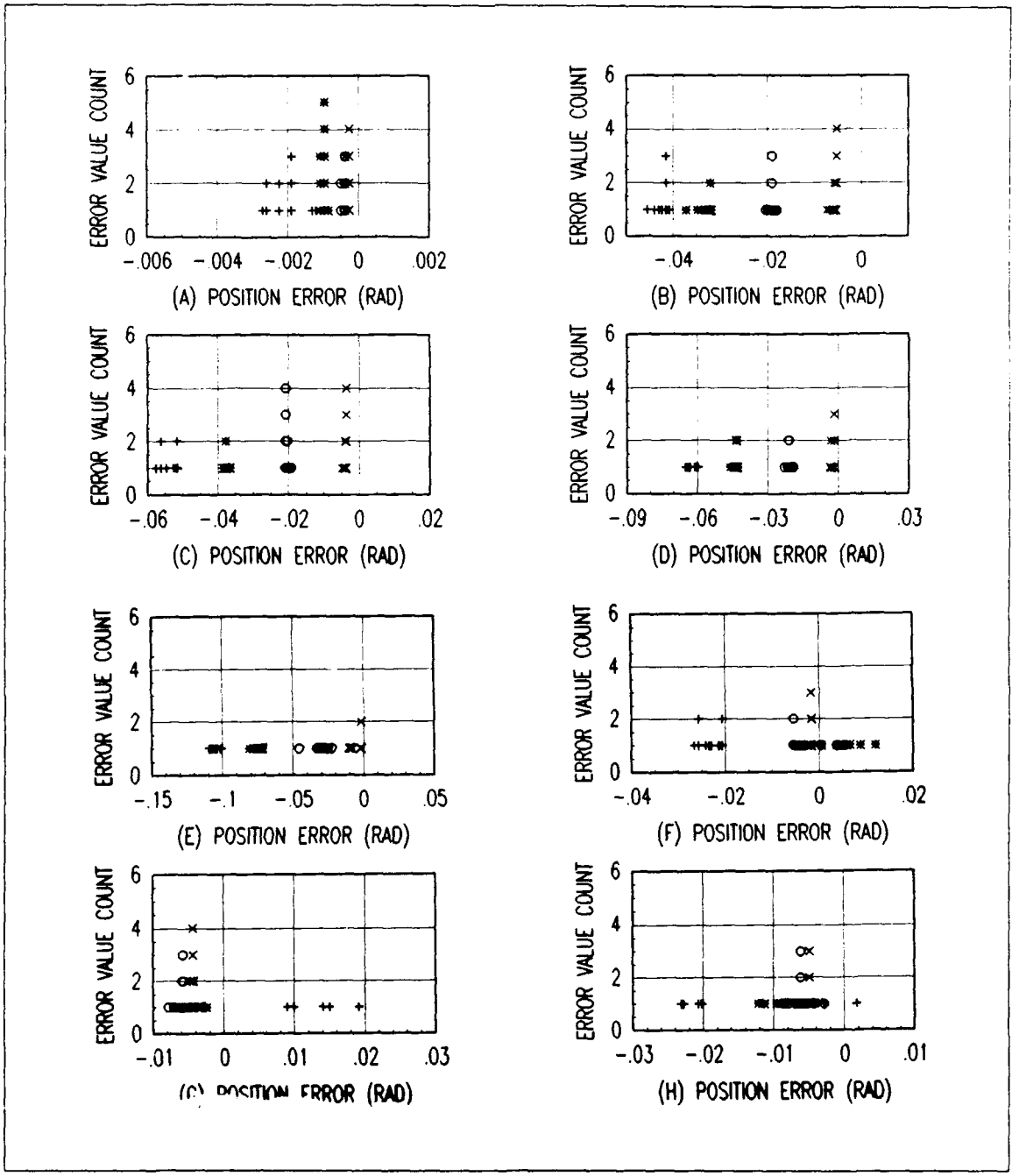

Figure 4.1. Payload Class Trajectory Data Dispersion

Trajectory Sample Period	10	50	100	150	200	250	300	330
Position Error Plot	A	B	C	D	E	F	G	H

×	Payload class zero errors	o	Payload class one errors
*	Payload class two errors	+	Payload class three errors

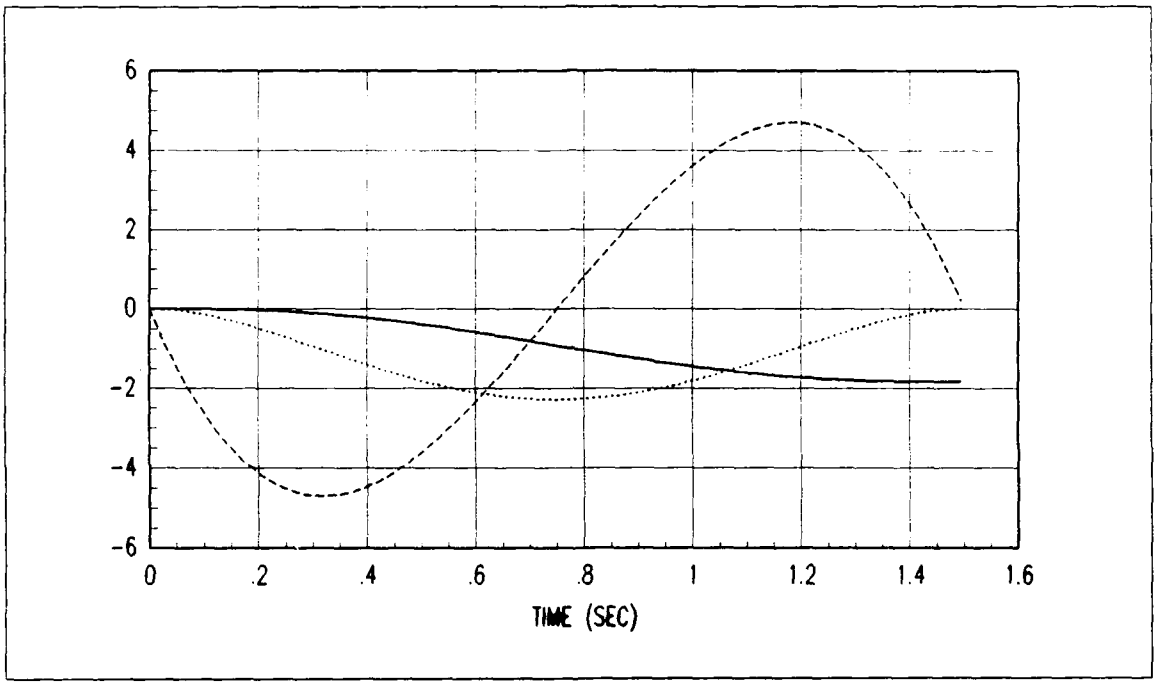

Figure 4.2. Original Trajectory Profiles

—	Position (rad)
···	Velocity (rad/sec)
- - -	Acceleration (rad/sec^2)

Figure 4.3. PUMA-560 with Payload Attached

Payload classes, representing zero to three kilograms in one kilogram increments, are indicated during training by a value of 0.9 for the actual class with all other classes set to 0.1. At specified training intervals (usually every 1000 training iterations) the nets are tested on their ability to detect the correct class from all training set vectors. Accuracy and error outputs are calculated using

$$Accuracy = \frac{n - e}{n} \tag{4.2}$$

$$Error = \frac{1}{2}\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(a_{ij} - d_{ij})^2}{n} \tag{4.3}$$

where

- n is the number of training vectors,
- e is the number of incorrect net class outputs,
- m is the number of net output nodes,
- a is the actual output node output, and
- d is the desired output node output.

The neural networks consist of (2) input nodes, (X) nodes per each of two hidden layers, and (4) output nodes where X denotes a number of nodes yet to be determined. The structures are noted as (2,X,X,4) for two hidden layer neural nets. The initial training sets consist of nine examplars from each payload class, or 36 examplars from a trajectory sample time. Based on an assumption that if the nets train with data representative of a worst case, they will train for all the other cases or sample times, 36 examplars from sample time 10 form the initial training set. A single hidden layer structure with 4 to 20 nodes in increments of 2 nodes is tried first. None of the single hidden layer structures began to train in 20000 training cycles. However, the first two hidden layer multilayer perceptron (MLP) neural network tried, starts to train in the first 3000 training iterations. Therefore, all remaining experiments focus on two hidden layer structures.

Searching for the acceptable number of hidden layer nodes to use entails testing nets of varying sizes for training rates, maximum accuracy, and minimum error. Neural nets with between 8 to 20 nodes per hidden layer are examined using training sets from time periods 10, 20, 30, 50, 100, and 150 of the trajectory. Figure 4.4 portrays the training time needed for the different net sizes to 'lock in' to the final trained state for a set of trajectory time periods. Figures 4.5 and 4.6 show the final accuracy and error achieved by the nets at sample periods 10, 20, 30, 50, 100, and 150. Testing results indicate that 12 and 16 node hidden layer nets work best with link three position information. Due to better performance from either 12 or 16 node nets in differing sample periods, one compromise in this development is to use MLP neural networks with a (2,14,14,4) structure.

The next set of tests used multiples of the standard deviation (σ) to determine the best value to use during normalization of the desired q_d and actual q trajectory data. Table 4.4 shows results from trajectory sample periods 10 and 50, and standard deviation multiples from 1/4 to 3. Using fractional values of the standard deviation yielded faster training and the same or poorer performance than using one sigma (1σ). In addition, larger sigma multiples gave very poor training performance. Therefore, one standard deviation is used for trajectory data normalization.

Training Accuracy (in percent)						
Sigma Multiple		$\sigma/4$	$\sigma/2$	σ	2σ	3σ
Sample Period	10	79.55	81.82	83.33	75.00	63.85
	50	100.00	100.00	100.00	61.00	—

Table 4.4. Standard Deviation Testing Results

In order to have nets trained on all available information before using them within the control system, a set of neural nets with one net per every ten sample periods is trained using 109 examplar vectors. However, in the trajectory error

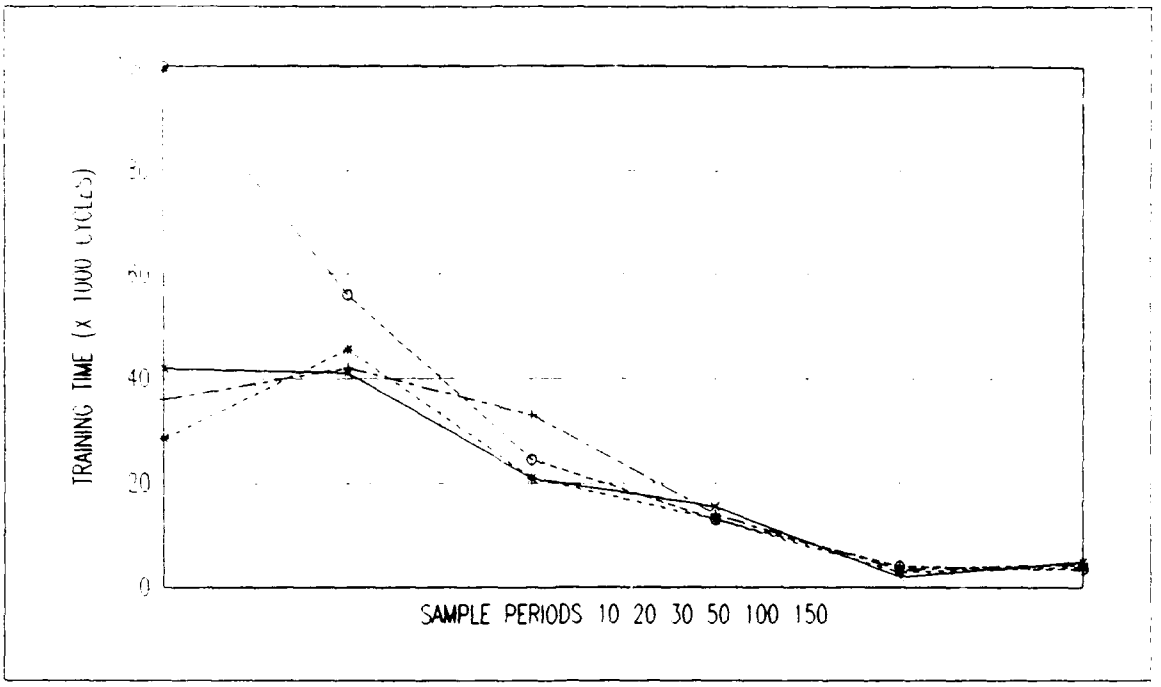

Figure 4.4. Training Time 'Lock In' Testing Results

×—×	8 node hidden layer nets
o- -o	12 node hidden layer nets
- -	16 node hidden layer nets
+- - -+	20 node hidden layer nets

data set there were more zero and one kilogram payload difference examplars than two or three kilogram variation examplars. To make up the 109 examplar vector data set, some additional examplars representing two and three kilogram payload variations are randomly added to the final training data set. One hundred and nine is the number of examplars in the set when I stopped adding examplars. Training nets with 109 examplar vectors takes an average of 400000 training iterations or 16 hours per net on a MicroVAX III.

In an effort to shorten the amount of training time a set of nets is trained using 44 examplar vector training sets. Figure 4.7 compares training of (2,14,14,4) nets with 44 versus 109 examplar vector training sets. Training time using the 44

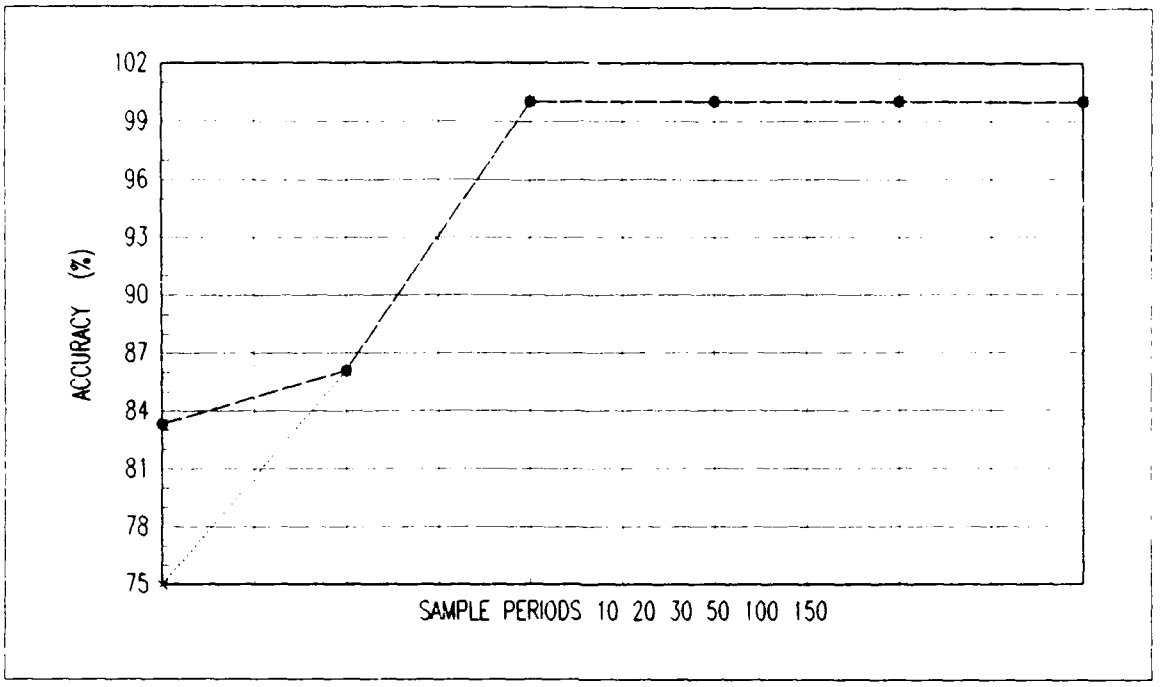

Figure 4.5. 'Lock In' Testing Training Accuracy

×—×	8 node hidden layer nets
o- -o	12 node hidden layer nets
- -	16 node hidden layer nets
+- - -+	20 node hidden layer nets

examplar vectors requires an average of 2 hours and 200000 training iterations per neural net. However, as shown in Figure 4.7, the final training accuracy is less and error is higher than training the nets with 109 training vectors. Therefore, nets trained for every ten sample periods using 109 examplar vectors are employed.

Neural networks trained with 109 examplars are tested in feedforward operation using 109 vectors of position information not previously seen by the networks. Accuracy and error are calculated the same as during training tests to allow for direct results comparison. Figure 4.8 shows the results from final training tests and feedforward operation tests. These test results indicate that neural networks

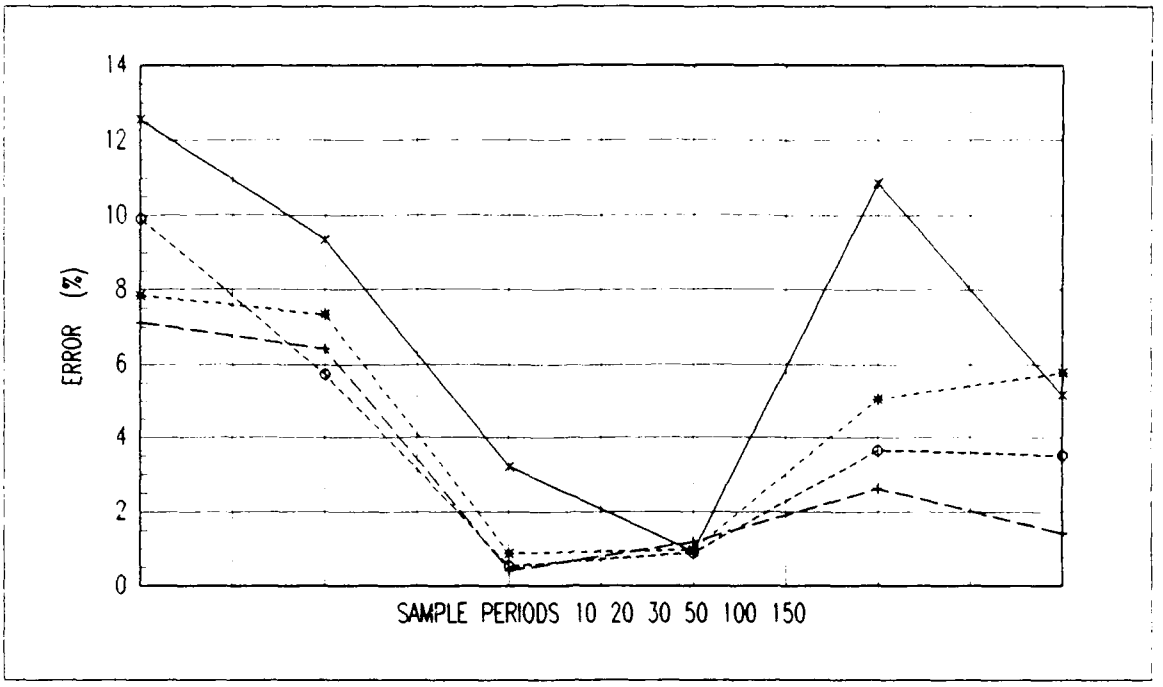

Figure 4.6. 'Lock In' Testing Training Error

×—×	8 node hidden layer nets
o– –o	12 node hidden layer nets
– –	16 node hidden layer nets
+– - –+	20 node hidden layer nets

can determine the payload mass parameter from trajectory error data.

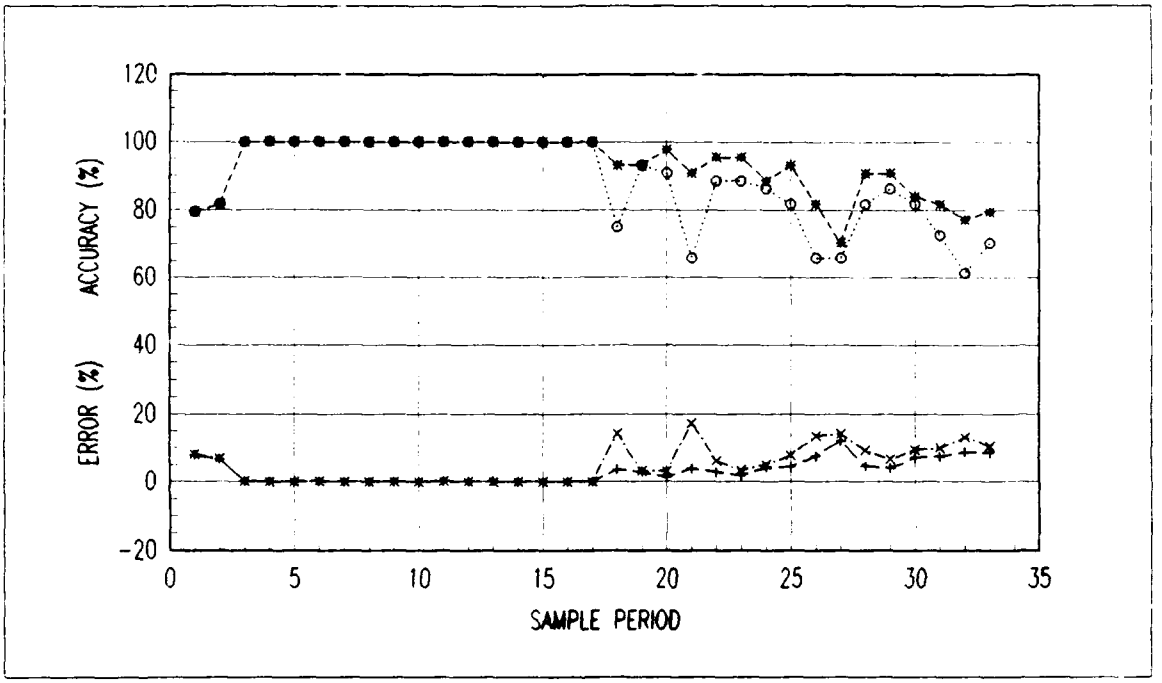

Figure 4.7. Training Results using 44 versus 109 Vector Training Sets

o···o	44 Vector Training Accuracy
×–·–×	44 Vector Training Error
– –	109 Vector Training Accuracy
+– · –+	109 Vector Training Error

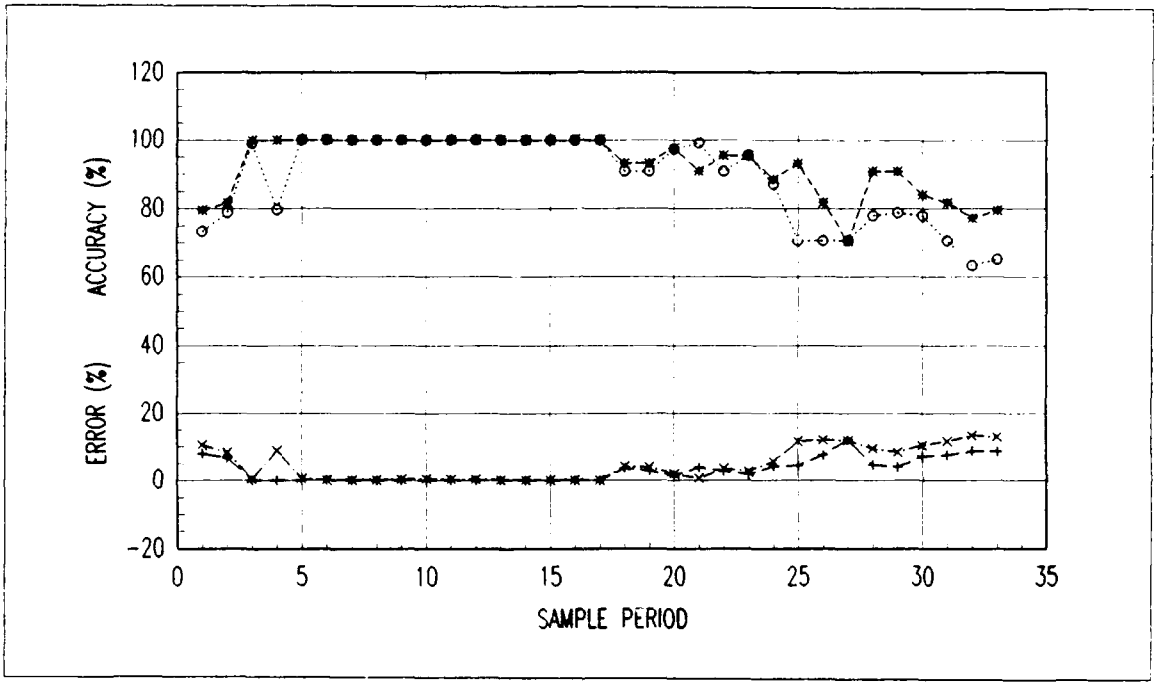

Figure 4.8. Final Training and Operation Testing Results

– –	Training Test Accuracy
+– · –+	Training Test Error
o···o	Operation Test Accuracy
×–·–×	Operation Test Error

4.4 Control System Implementation

Once adequately trained nets are available, the next task is to set up the necessary tools to use them within the adaptive model-based control structure. First, the framework for loading the nets is established and tested to make sure the proper values are correctly placed in the proper neural network array. Next, the unnormalized position data, provided by the joint encoders, is used to verify the feedforward operation in simulated realtime operation of the neural net arrays. Finally, the recomputation of the dynamic feedforward compensation based upon the mass estimate is verified.

Once both the loading and feedforward operation mechanisms are working correctly, they are put together and tested in simulated realtime operation using a set of test examplars. Next, they are integrated into an existing robot control simulation package to find and correct any undiscovered problems. Ultimately, the routines are integrated into the operational structure (AMBNNC) outlined in Chapter 3 for on-line experimentation.

4.5 Experimental Performance Evaluation

The following experimental results are chosen to demonstrate the performance potential of the AMBNNC as a direct adaptive control technique. A compilation of results that add substance to the following discussion are located in an internal report along with the listings of software tools developed as part of the project (see Technical Report ARSL-89-12). In addition, results using the compromise for updating the system dynamics mentioned in Chapter 3 are noted with an asterisk (*) in the plot legend. This is due to the compromise being investigated and developed late in the testing regimen. Most results use dynamic updates only when a change other than zero or the current payload mass value occurs. The case where the compromise improves performance is when there is no payload; otherwise, performance deviations are not apparent. All trajectories used during the

experiments are described in Table 4.3. Figure 4.9 illustrates trajectory initial positions A - D and the paths followed by trajectories one, two, and three. Also, each trajectory, initial position, and payload variation found in Tables 4.2 and 4.3 are performed ten times to check performance repeatability. The presentation begins by looking at the AMBNNC trajectory tracking performance using the trajectory for which the neural nets are trained.

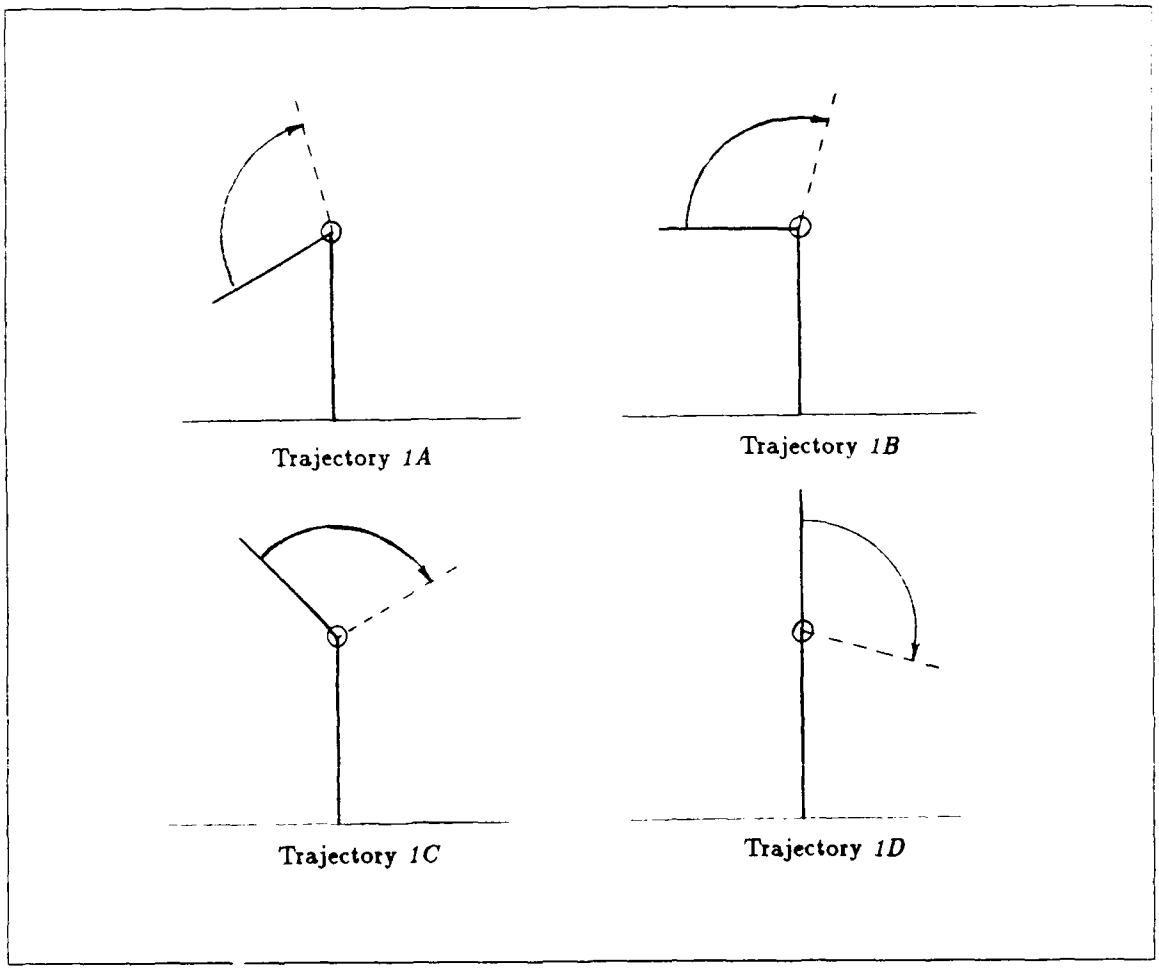

Figure 4.9. Trajectory Profiles

4.5.1 Original Trajectory Performance Trajectory *1A* is the trajectory used to train the neural networks. Figures 4.10 - 4.13 show the trajectory tracking performance of the AMBNNC versus SMBC with payloads of zero, one, two, and three kilograms for trajectory *1A*. The no payload case is considered difficult to handle [44], yet the AMBNNC performance closely follows the SMBC performance and is better in some portions of the trajectory. In each of the error plots for the one, two and, three kilogram payload situations the controller is initially informed there is zero kilograms or no payload. The plots reveal that AMBNNC performance is superior to the SMBC, by at least two times for a three kilogram payload and six times for a one kilogram payload, when both are without *a priori* payload information. Also, the AMBNNC without payload knowledge achieves similar or better trajectory tracking than the SMBC with the correct payload information.

In all of these tests the neural nets correctly identified the payload within three neural network sample periods. Remember that the first neural net sample is taken at 45 milliseconds into the 1.5 second trajectory. Full AMBNNC adaptation to the correct payload is made at the next servo sample period with trajectory correction occurring by the next net sample period. Examples showing NNPE estimates and how quickly the AMBNNC affects trajectory tracking are shown and described in later sections. These results verify the ability of the neural networks to provide accurate payload estimates during high speed manipulator motion. In addition to providing payload estimates, the results indicate an aptitude for compensating for unmodeled dynamics.

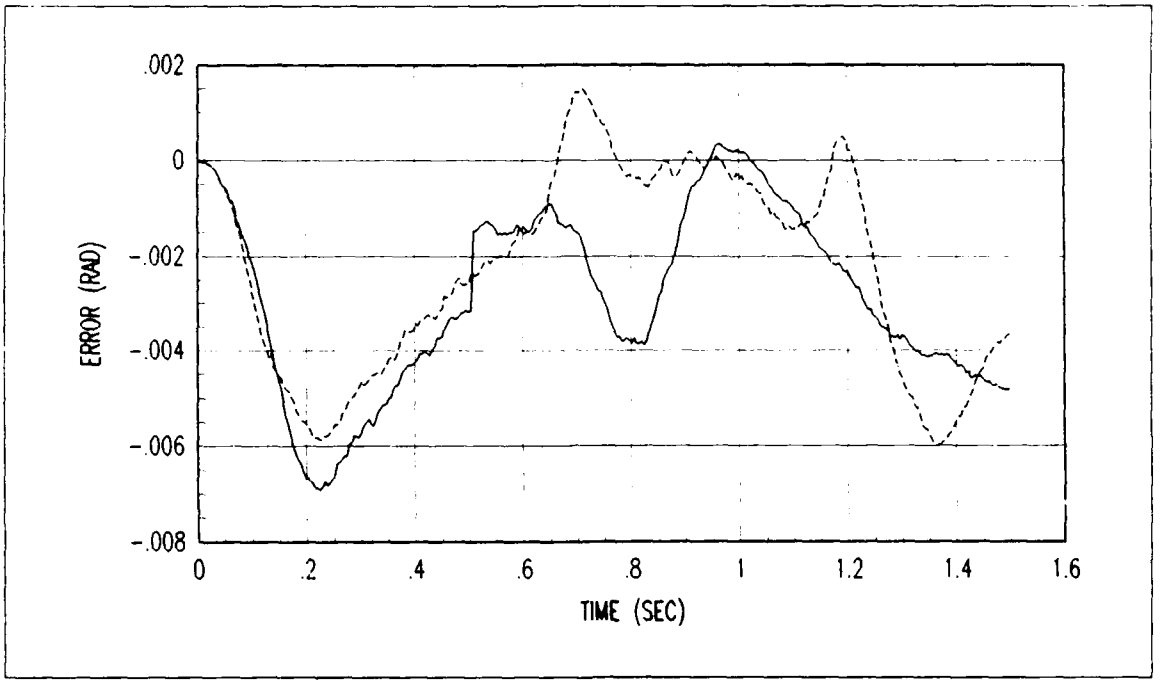

Figure 4.10. Tracking Accuracy using NNPE on Trajectory 1A with 0.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

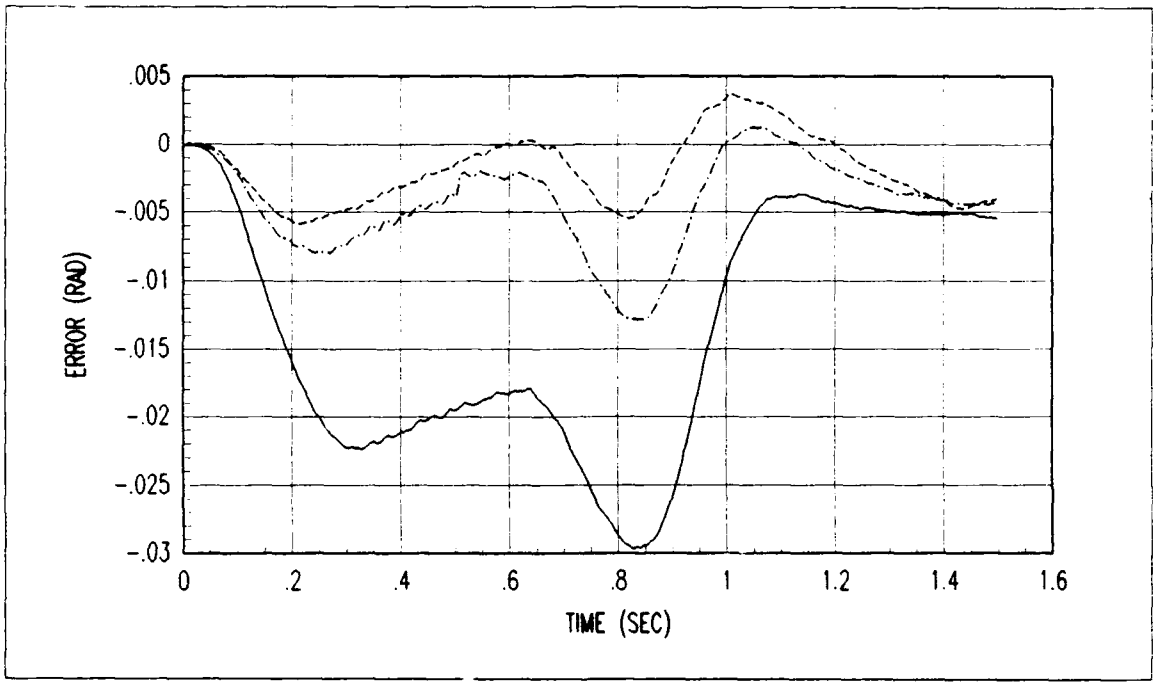

Figure 4.11. Tracking Accuracy using NNPE on Trajectory 1A with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
– · –	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

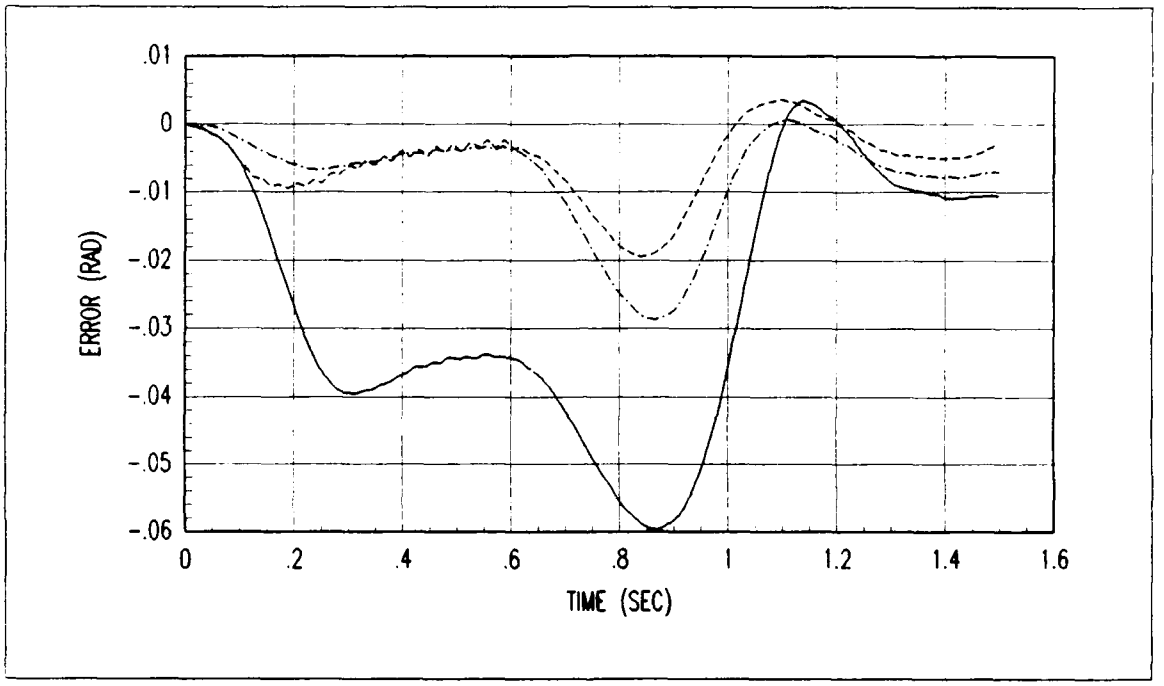

Figure 4.12. Tracking Accuracy using NNPE on Trajectory 1A with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- · -	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

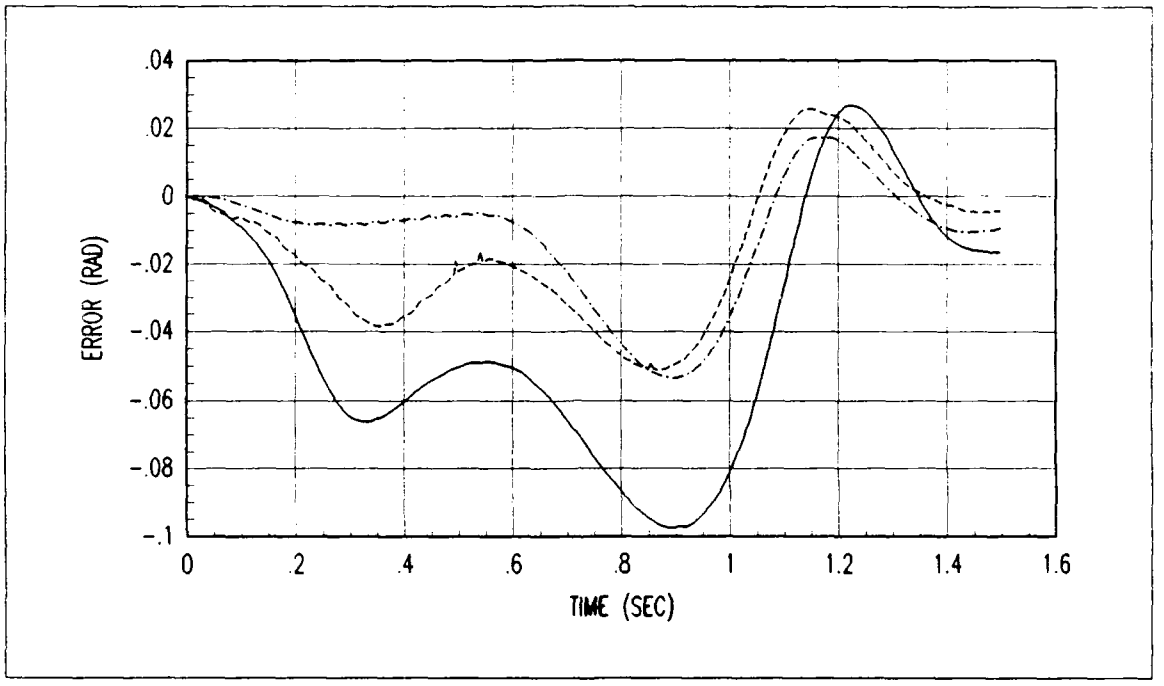

Figure 4.13. Tracking Accuracy using NNPE on Trajectory 1A with 3.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- · -	SMBC w/3.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

4.5.2 Performance on Alternative Trajectories To further test limitations of the AMBNNC, experiments are performed using trajectories for which the neural nets are *not* trained. These tests include trajectories using the same motion and speed as the original trajectory; however, they use different initial positions. The tests are thereby able to test the neural network's ability to generalize the payload from trajectory errors occurring during manipulator exposure to degrees of gravity and inertia not experienced in the original trajectory.

Figures 4.14 and 4.15 are chosen to represent the tracking performance of the AMBNNC versus the SMBC on trajectory *1B* with one and two kilogram payloads. For both payloads the AMBNNC out performs the SMBC. The next two figures focus on the AMBNNC execution of trajectories *1C* and *1D*. AMBNNC performance in Figure 4.16 closely parallels the SMBC performance when the SMBC is given the correct payload information. However, when both the AMBNNC and SMBC are initially told the payload is zero kilograms, the AMBNNC performance is far superior to the SMBC performance. Figure 4.17 illuminates the problem in payload detection when the manipulator is essentially falling into the gravity field throughout the entire trajectory. Due to the small amount of initial excitation, trajectory *1D* is the hardest trajectory for any payload estimator to detect payload differences. As is shown during neural network 'firing' evaluations, the nets do not identify the payload until near the end of the arm motion. Probable solutions are to train the nets to detect negative payload variations or give the nets additional training on paths similar to trajectory *1D*.

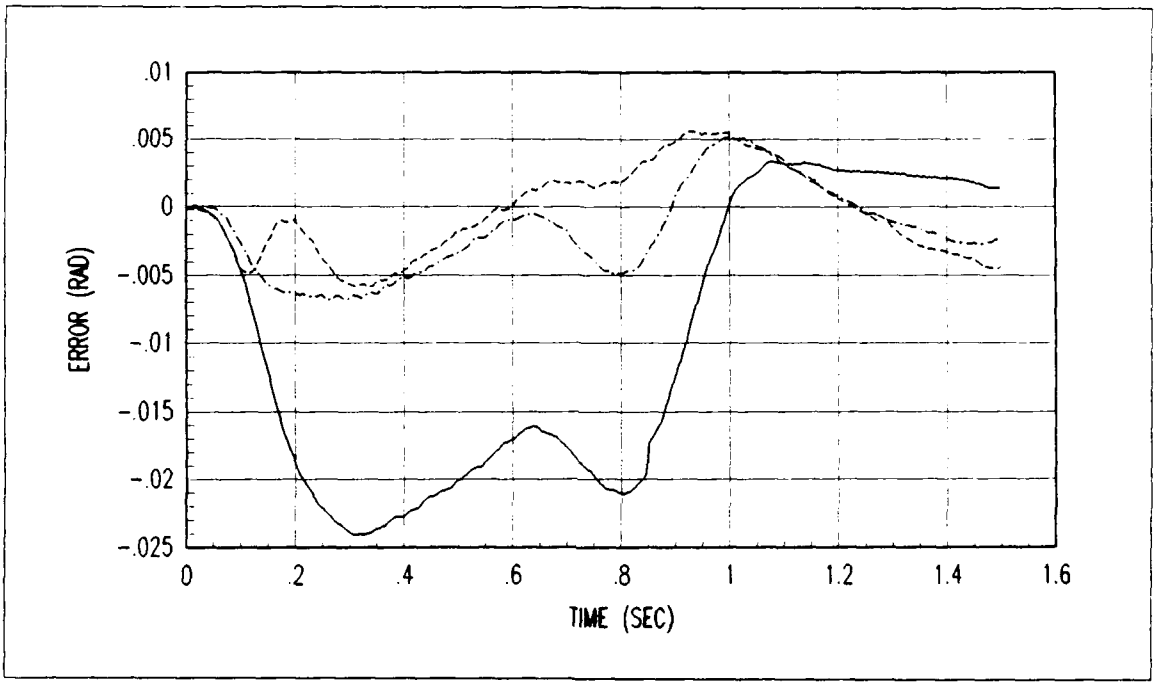

Figure 4.14. Tracking Accuracy using NNPE on Trajectory 1B with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- · -	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

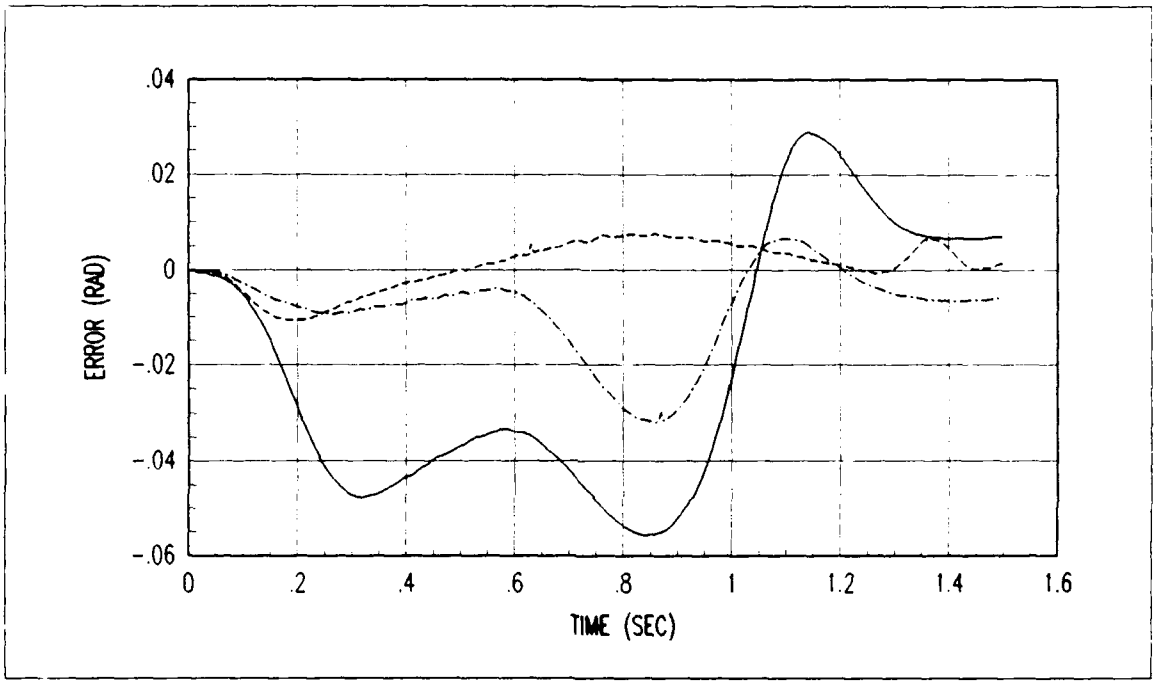

Figure 4.15. Tracking Accuracy using NNPE on Trajectory 1B with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

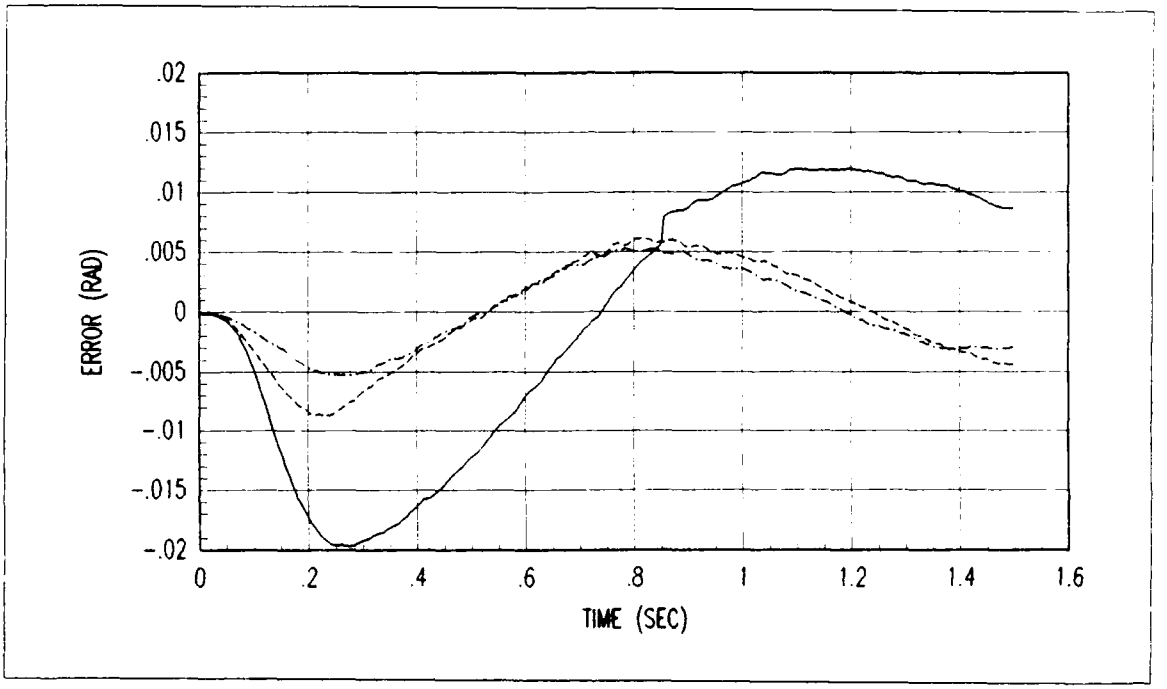

Figure 4.16. Tracking Accuracy using NNPE on Trajectory 1C with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
– · –	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

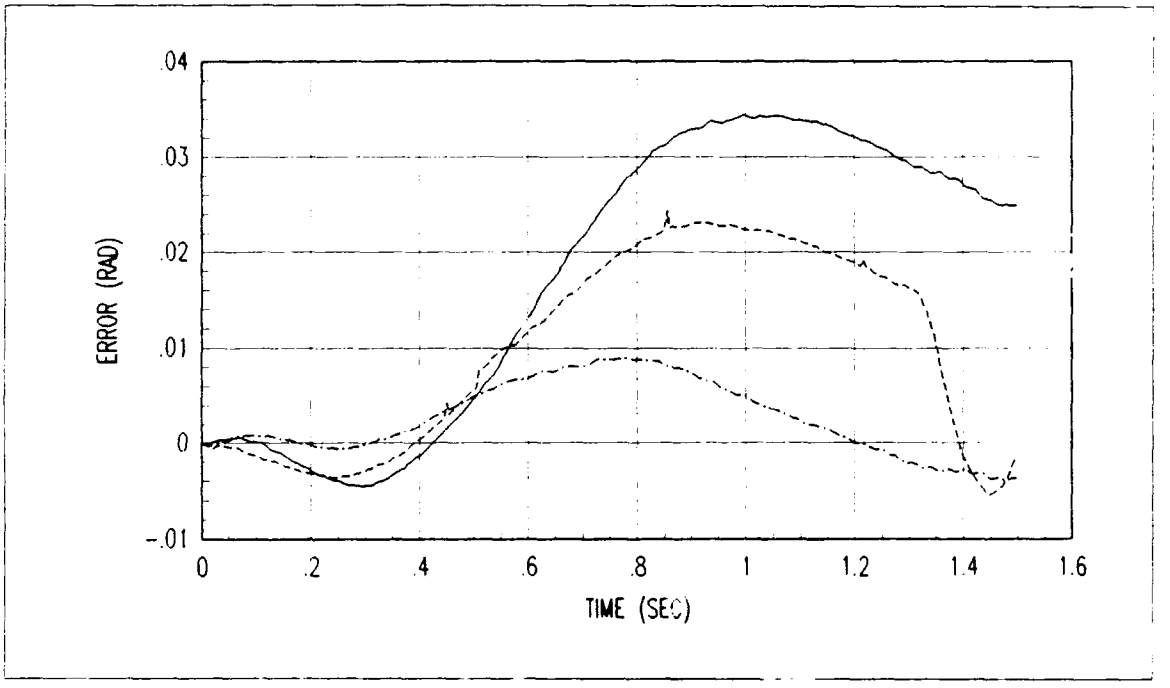

Figure 4.17. Tracking Accuracy using NNPE on Trajectory 1D with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

4.5.3 Performance on Variable Length Trajectories As another indication of the potential versatility of the AMBNNC, Figure 4.18 shows the tracking of the AMBNNC versus the SMBC on trajectory *5A*. Trajectory *5A* moves half the distance in the same time as trajectory one. With a one kilogram payload the tracking accuracy is better than the SMBC with the same initial payload information. Nonetheless, over most of the trajectory the SMBC performs better than the AMBNNC when the SMBC is given correct payload information. Figures 4.19 and 4.20 show AMBNNC tracking versus SMBC tracking on trajectories *2A* and *3A*, respectively. Overall performance of the AMBNNC with incorrect payload information is superior to the SMBC performance irregardless of the payload information given to the SMBC. Similar tests on other trajectories and payload conditions show similar performance characteristics. The neural net sample rate is the primary factor that changes for these trajectories and is the subject of the next discussion.

4.5.4 Neural Network Sample Rate Variation Tests Tests on trajectories two through five indicate a tracking accuracy dependence on how often the neural networks are sampled during manipulator motion. To investigate the tracking accuracy dependence on sample rate, a two kilogram payload is used on trajectory *1A*. Figure 4.21 shows a set of single runs with sample rates of 4.5, 5.4, 6.3, and 7.2 milliseconds (ms) versus the SMBC using a 4.5 ms sample rate. These rates represent neural network sample rates of 45, 54, 63, and 72 ms. AMBNNC performance with larger sample periods is better than the SMBC with the same payload information. Note that the servo sample rates are changed during these tests which cause some manipulator performance degradation to be included in the tracking error. With two kilograms payload and a neural network sample rate of 63 ms, Figure 4.22 shows that AMBNNC performance for the 63 ms sample rate is consistent for various initial mass information conditions. Performance consistency, even during poor performance, is one trait of the AMBNNC.

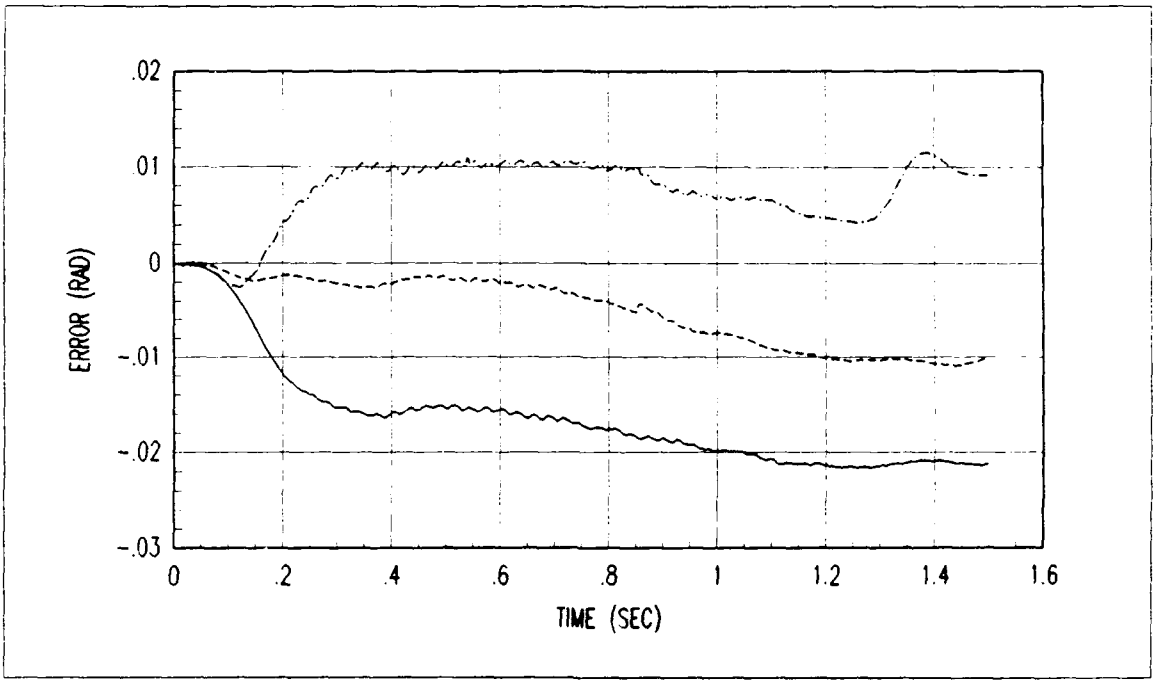

Figure 4.18. Tracking Accuracy using NNPE on Trajectory 5A with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	SMBC w/1.0 Kg Load information
– · –	AMBNNC w/0.0 Kg Load information

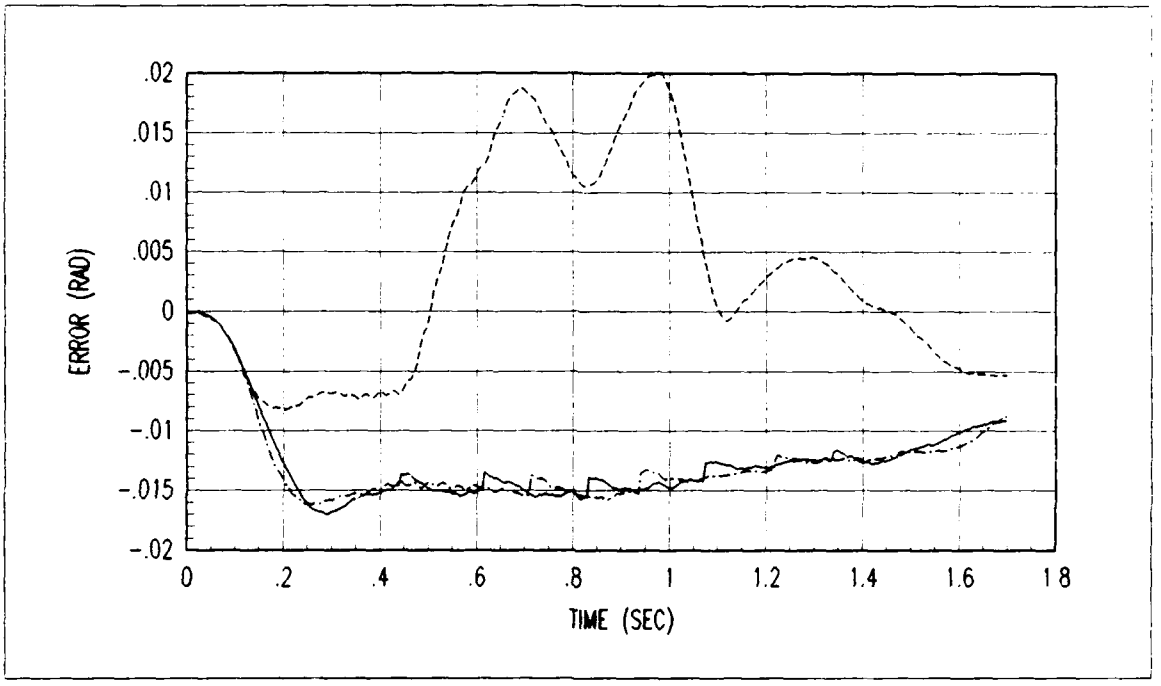

Figure 4.19. Tracking Accuracy using NNPE on Trajectory 2A with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- · -	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

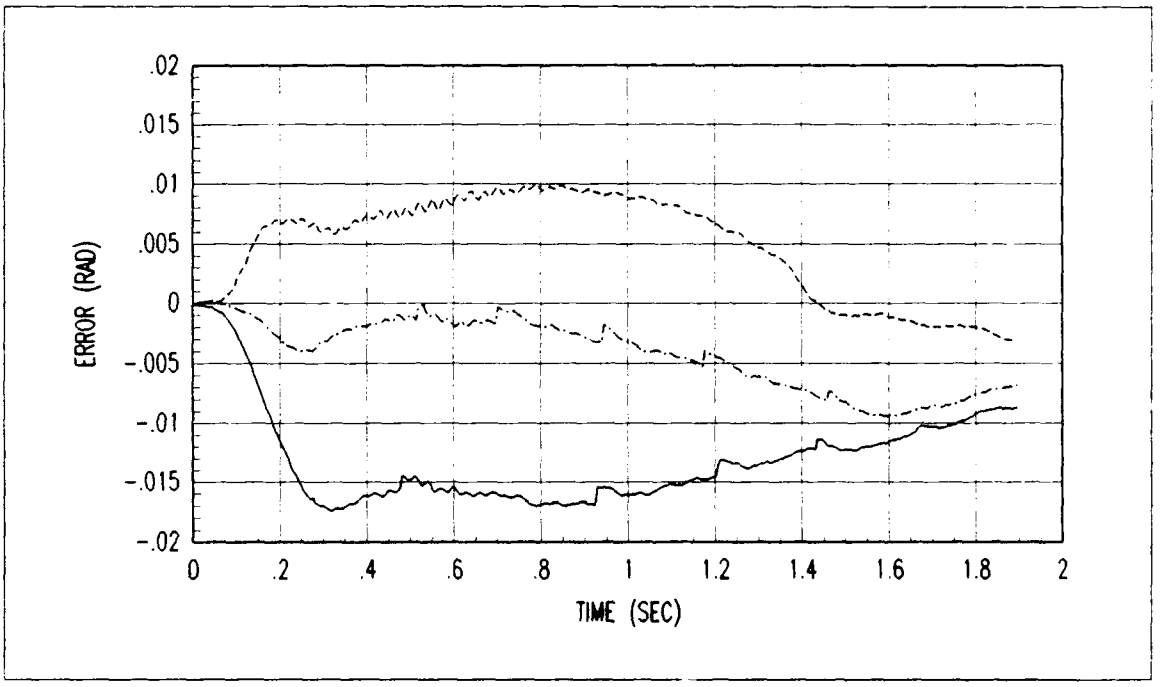

Figure 4.20. Tracking Accuracy using NNPE on Trajectory 3A with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
– · –	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information

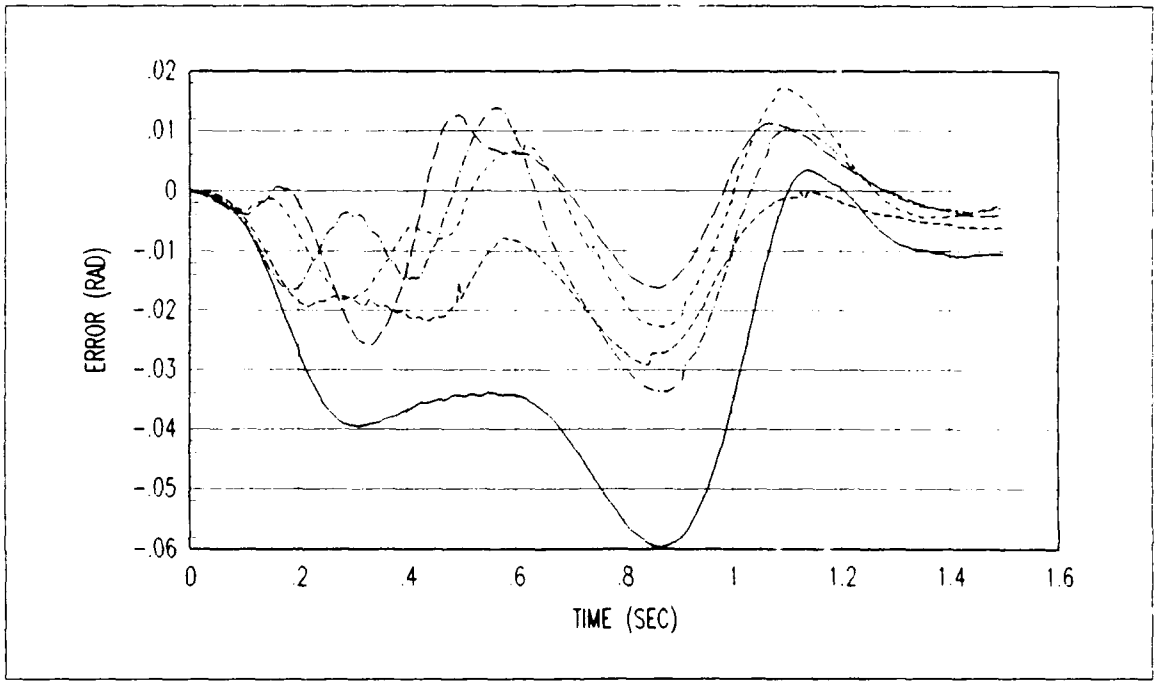

Figure 4.21. Tracking Accuracy using NNPE on Trajectory 1A with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information and 45 ms neural net sample rate
- · -	AMBNNC w/0.0 Kg Load information and 54 ms neural net sample rate
- - -	AMBNNC w/0.0 Kg Load information and 63 ms neural net sample rate
— —	AMBNNC w/0.0 Kg Load information and 72 ms neural net sample rate

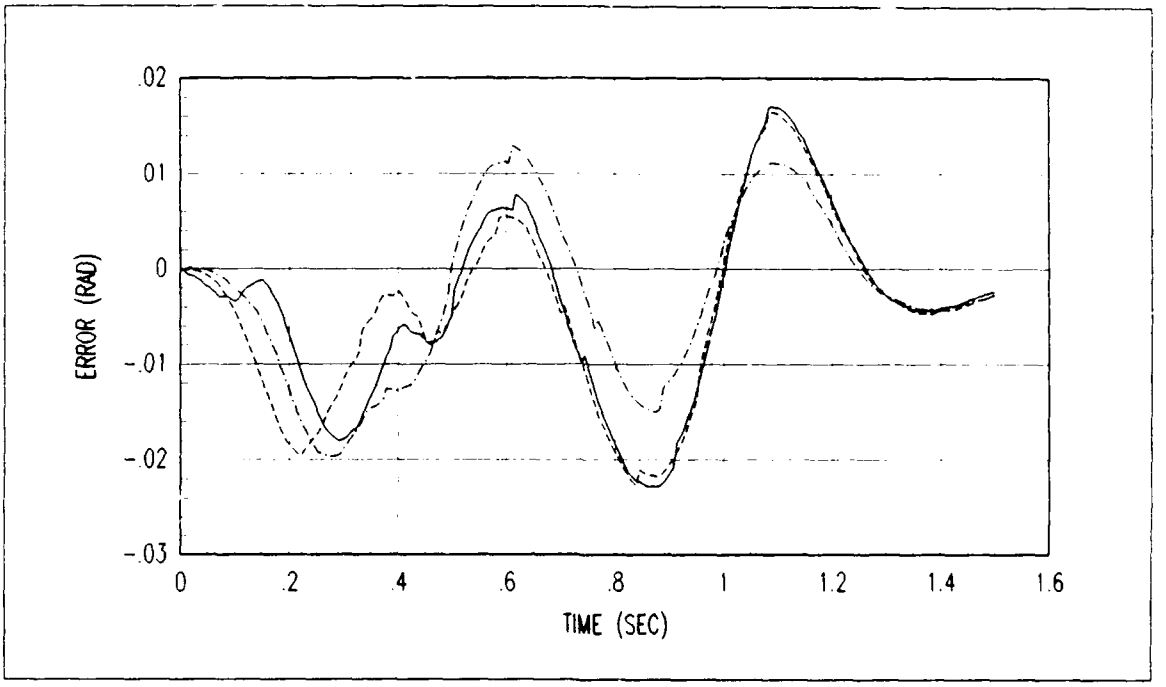

Figure 4.22. Tracking Accuracy using NNPE on Trajectory 1A with 2.0 Kilogram Payload

—	AMBNNC w/0.0 Kg Load information
- - -	AMBNNC w/1.0 Kg Load information
– · –	AMBNNC w/2.0 Kg Load information

4.5.5 Performance Repeatability Tests Figures 4.23 - 4.26 exhibit the mean and +/- one sigma for ten runs of trajectories *1A - 1D*. The payload is one kilogram and the AMBNNC is initially informed that the payload is zero kilograms. Five runs are made with a calibration prior to execution. Using the last calibration of the previous five runs, five additional runs are made with a return to the initial position between the runs. For comparison the plots include the SMBC tracking accuracy for the same conditions given the AMBNNC and also when the SMBC is told the correct payload. AMBNNC adaptation is easily seen in the convergence of the standard deviation in each plot.

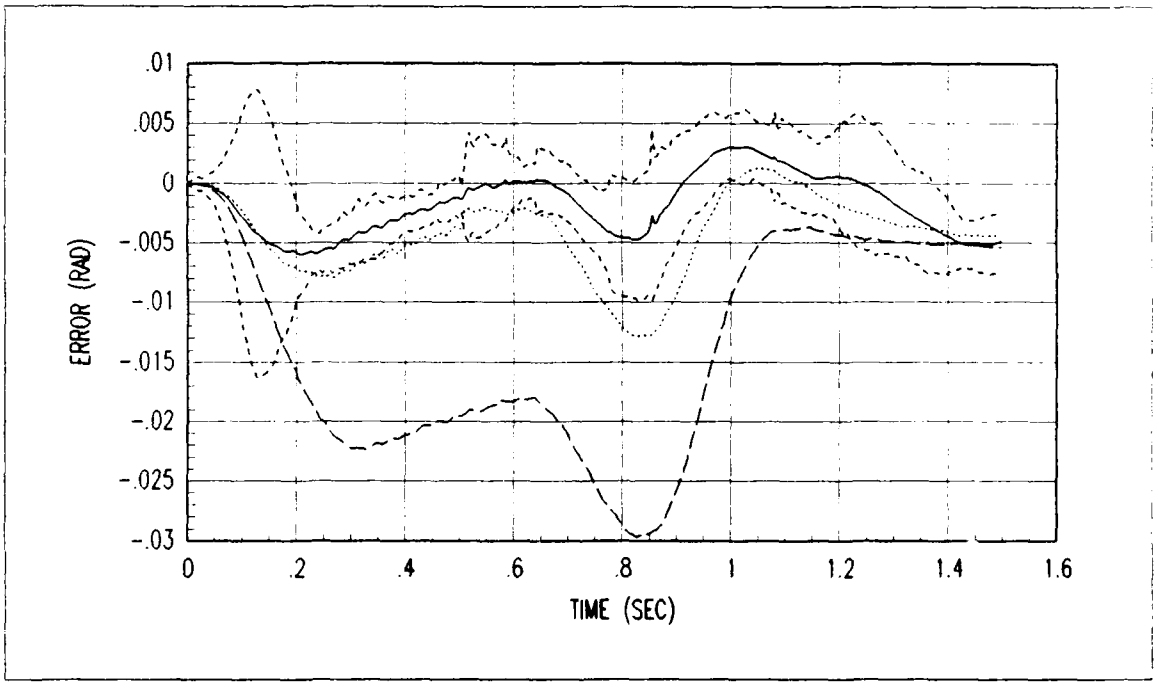

Figure 4.23. Ten Run Mean Tracking Accuracy using NNPE on Trajectory 1A with 1.0 Kilogram Payload

- -	SMBC w/0.0 Kg Load information
···	SMBC w/1.0 Kg Load information
—	AMBNNC w/0.0 Kg Load information
- - -	+/- one standard deviation

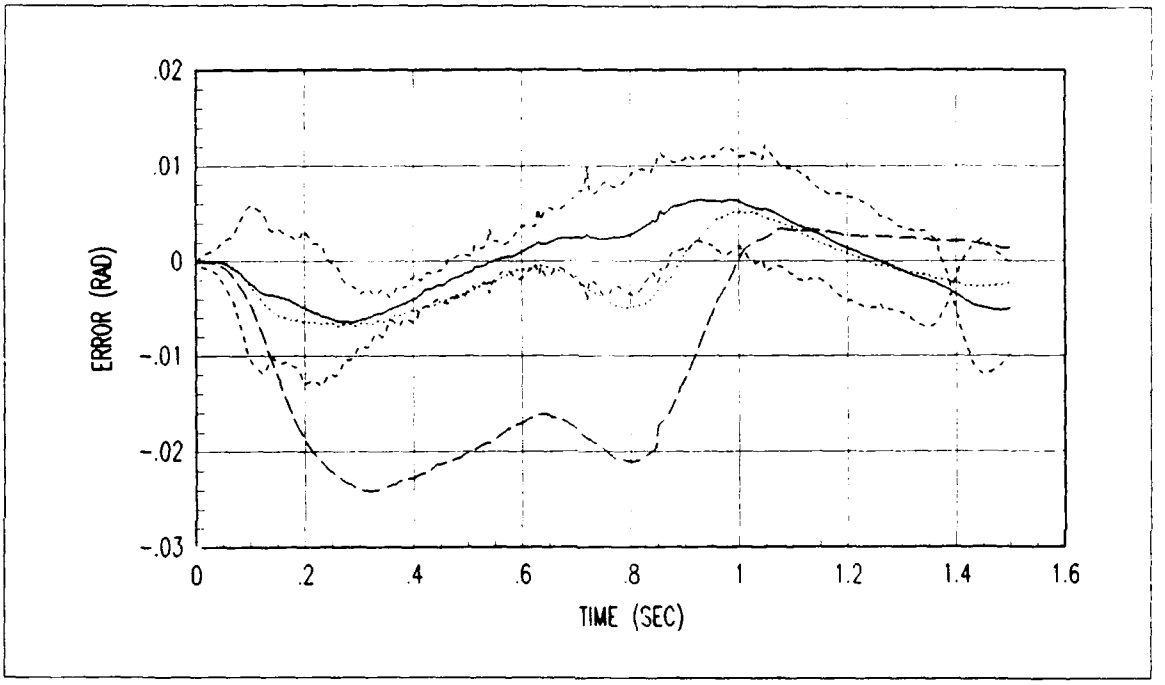

Figure 4.24. Ten Run Mean Tracking Accuracy using NNPE on Trajectory 1B with 1.0 Kilogram Payload

- -	SMBC w/0.0 Kg Load information
···	SMBC w/1.0 Kg Load information
—	AMBNNC w/0.0 Kg Load information
- - -	+/- one standard deviation

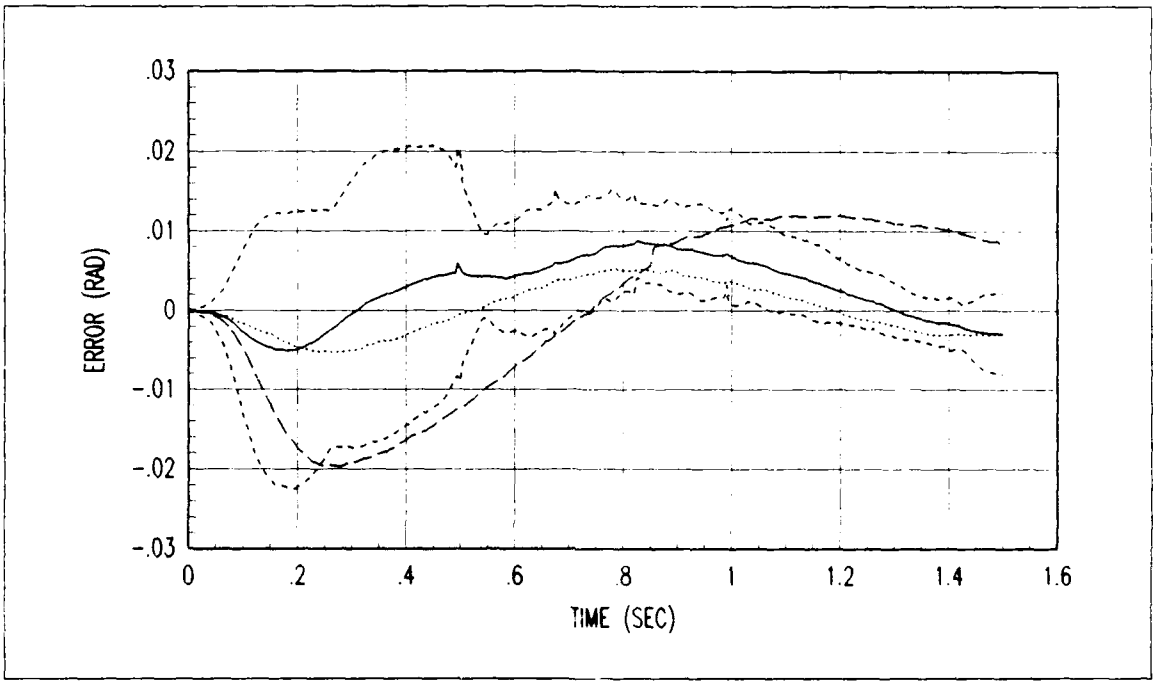

Figure 4.25. Ten Run Mean Tracking Accuracy using NNPE on Trajectory 1C with 1.0 Kilogram Payload

- -	SMBC w/0.0 Kg Load information
···	SMBC w/1.0 Kg Load information
—	AMBNNC w/0.0 Kg Load information
- - -	+/- one standard deviation

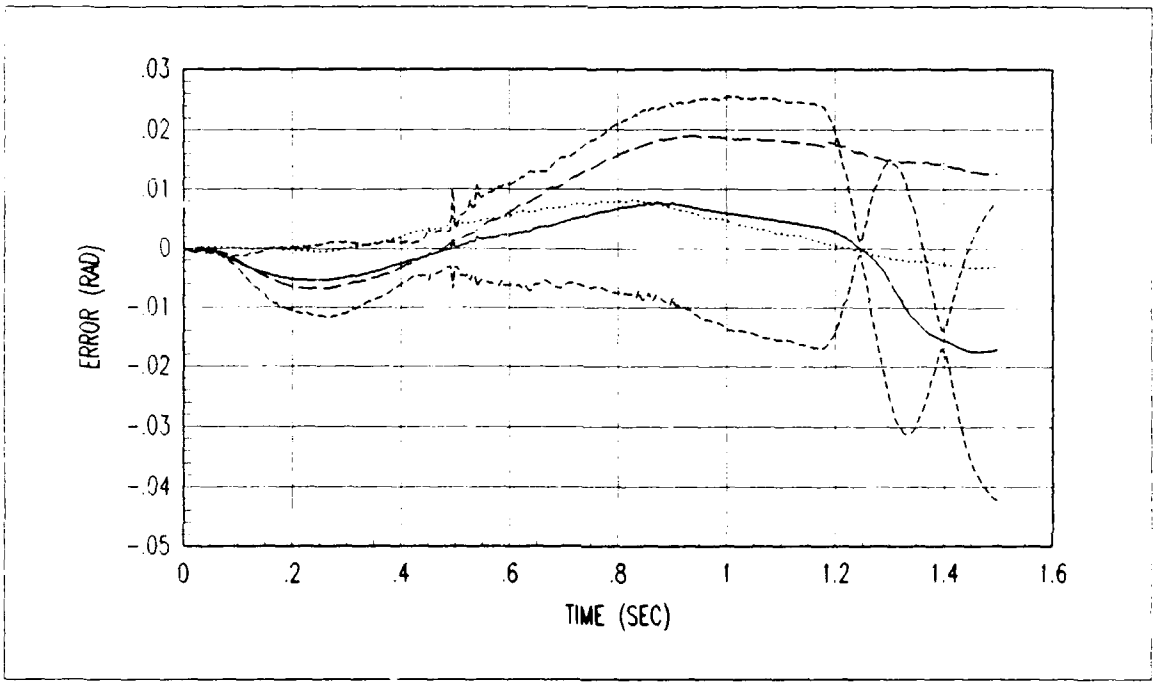

Figure 4.26. Ten Run Mean Tracking Accuracy using NNPE on Trajectory 1D with 1.0 Kilogram Payload

- -	SMBC w/0.0 Kg Load information
···	SMBC w/1.0 Kg Load information
—	AMBNNC w/0.0 Kg Load information
- - -	+/- one standard deviation

4.5.6 Payload Range Performance Tests As another probe into the AMBNNC performance envelope, sets of individual runs are performed using a range of payload values outside the scope of payload mass variations with which the neural networks are trained. Figures 4.27 and 4.28 display AMBNNC execution for initial payload information from zero through five kilograms with an actual payload of two kilograms on trajectories *1A* and *1C*. These tests introduce negative mass variations and the neural nets used in these experiments are only trained to detect positive mass changes. In spite of the lack of proper training, the AMBNNC is able to force trajectory tracking convergence before the end of the trajectory. An example of the neural net 'firing' patterns which bring about the convergence is found in the next segment.

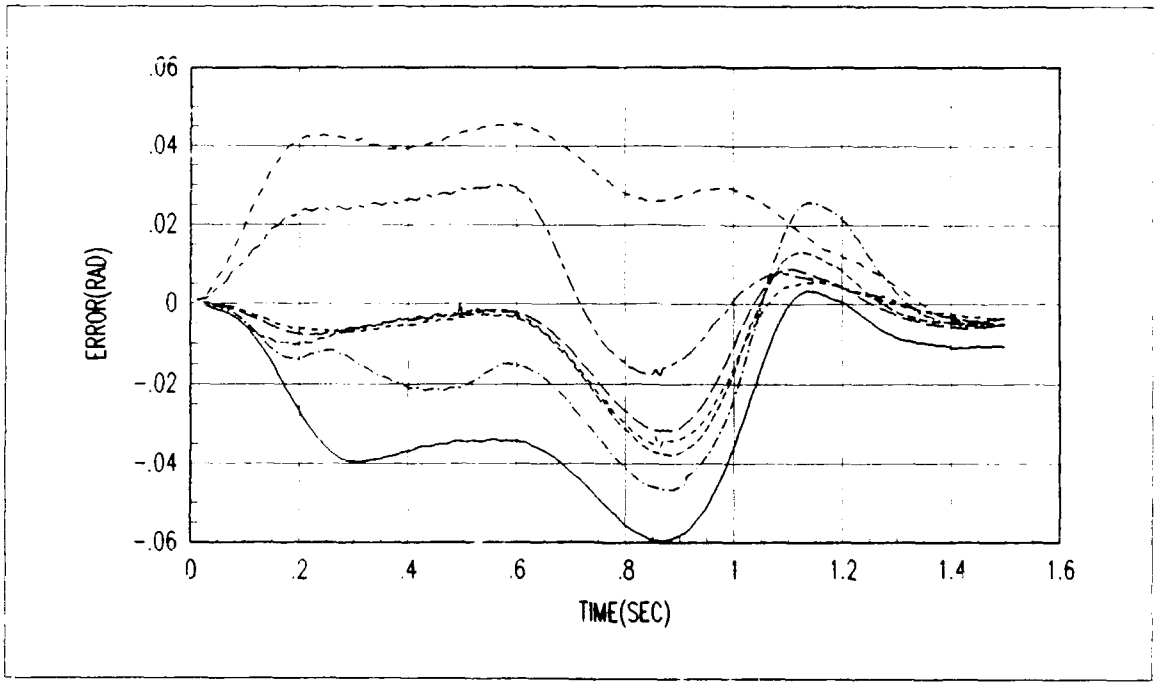

Figure 4.27. Tracking Accuracy using NNPE on Trajectory 1A with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
- · -	AMBNNC w/1.0 Kg Load information
— — —	AMBNNC w/2.0 Kg Load information
- - -	AMBNNC w/3.0 Kg Load information
— · —	AMBNNC w/4.0 Kg Load information
- -	AMBNNC w/5.0 Kg Load information

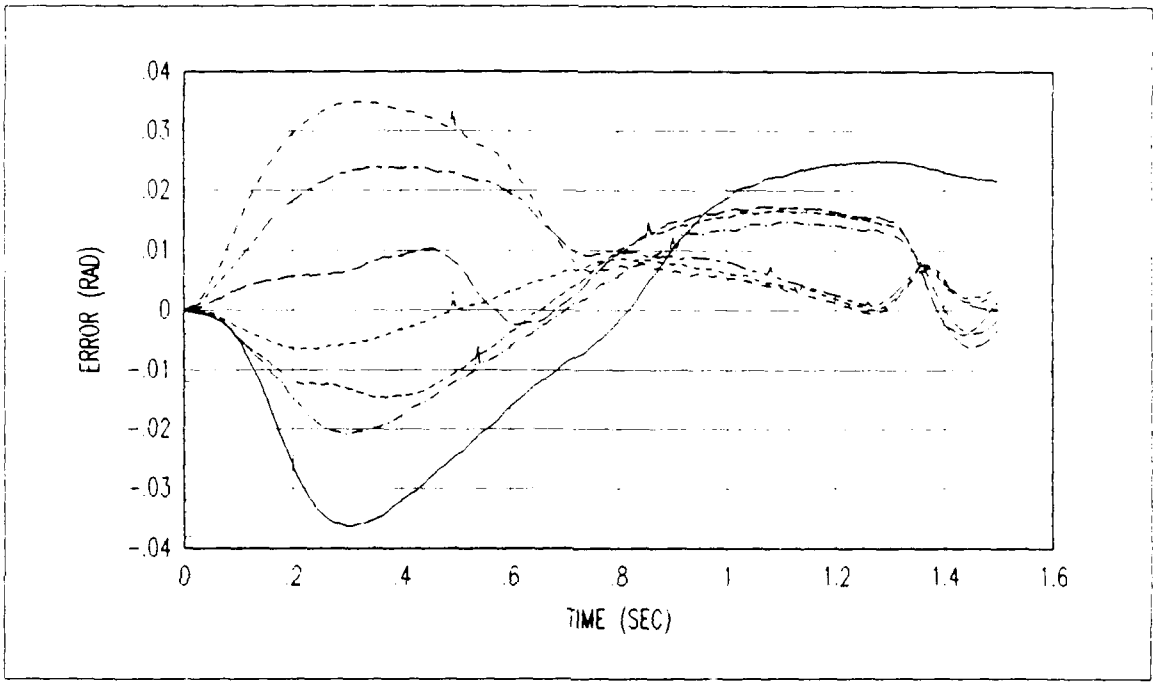

Figure 4.28. Tracking Accuracy using NNPE on Trajectory 1C with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
- · -	AMBNNC w/1.0 Kg Load information
· · ·	AMBNNC w/2.0 Kg Load information
— — —	AMBNNC w/3.0 Kg Load information
— · —	AMBNNC w/4.0 Kg Load information
- -	AMBNNC w/5.0 Kg Load information

4.5.7 Neural Network 'Firing' during Test Execution The ability of the AMBNNC to improve the trajectory tracking performance is directly a result of the neural networks being able to quickly and accurately determine the payload mass parameter. The following presentation clearly demonstrates that the neural networks are able to provide fast and accurate payload mass estimates during high speed manipulator motion. Two noted exceptions are when trajectory *1D* is being traversed and when the AMBNNC is originally told that the mass is greater than is actually attached. Keep in mind that unless an asterisk is in the plot legend, the payload mass parameter is changed only when a new value other than zero or the previous estimate is given by the neural nets. For those plots with an asterisk, the system dynamics are updated with each payload estimate through the first 450 ms of a 1.5 second trajectory.

4.5.7.1 Original Trajectory The first four plots (Figures 4.29 - 4.32) illustrate the neural net payload estimates for zero, one, two, and three kilogram payloads on trajectory *1A*. Figure 4.29 portrays the zero kilogram payload case using the compromise for system dynamics updating. Remember that the compromise method updates the system dynamics with every payload estimate through sample time 100 (450 ms). Beginning at sample time 110 (495 ms) and continuing for the duration of the trajectory, the payload is updated only when a variation in the payload is detected. Using the compromise approach, the first estimate other than zero comes at 630 ms into the trajectory and changes the payload mass parameter to one kilogram. The mass parameter remains at one kilogram until 1.12 seconds when it is changed to three kilograms. At about 1.35 seconds it is changed to two kilograms and finally to one kilogram at about 1.44 seconds. The behavior of the neural networks for the case just described is typical for every trajectory execution.

Figure 4.30 shows the payload estimates for trajectory *1A* with a one kilogram payload. Not using the compromise for system updates, the correct payload value of

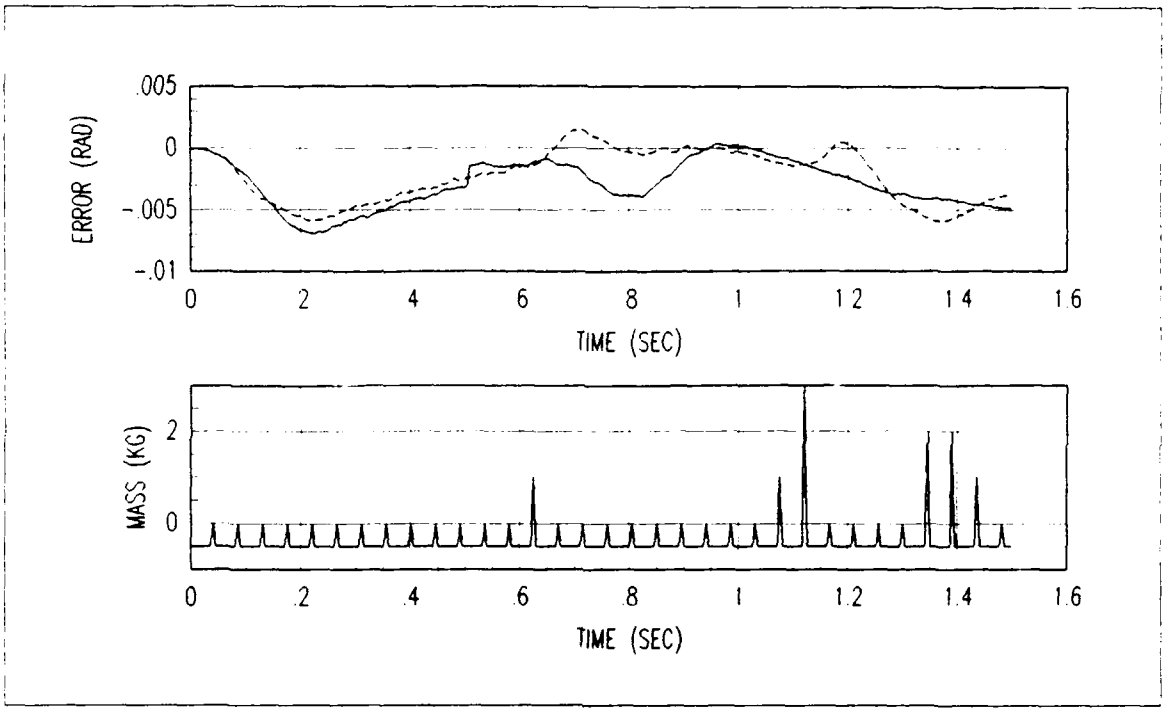

Figure 4.29. Firing of Neural Networks during Trajectory 1A with 0.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs (*)

one kilogram is achieved on the third neural net trajectory tracking error sampling. The payload mass parameter remains unchanged throughout the trajectory except for a change to two kilograms at 1.44 seconds. Similarly the two kilogram payload is identified by the third ANN trajectory error sampling as exhibited in Figure 4.31. The payload mass parameter is again changed to various values towards the end of the trajectory. The compromise update technique is used for the last time in the three kilogram payload run shown in Figure 4.32. The payload is immediately detected by the neural nets, and the 'firings' of the nets indicate how closely the desired trajectory is being followed.

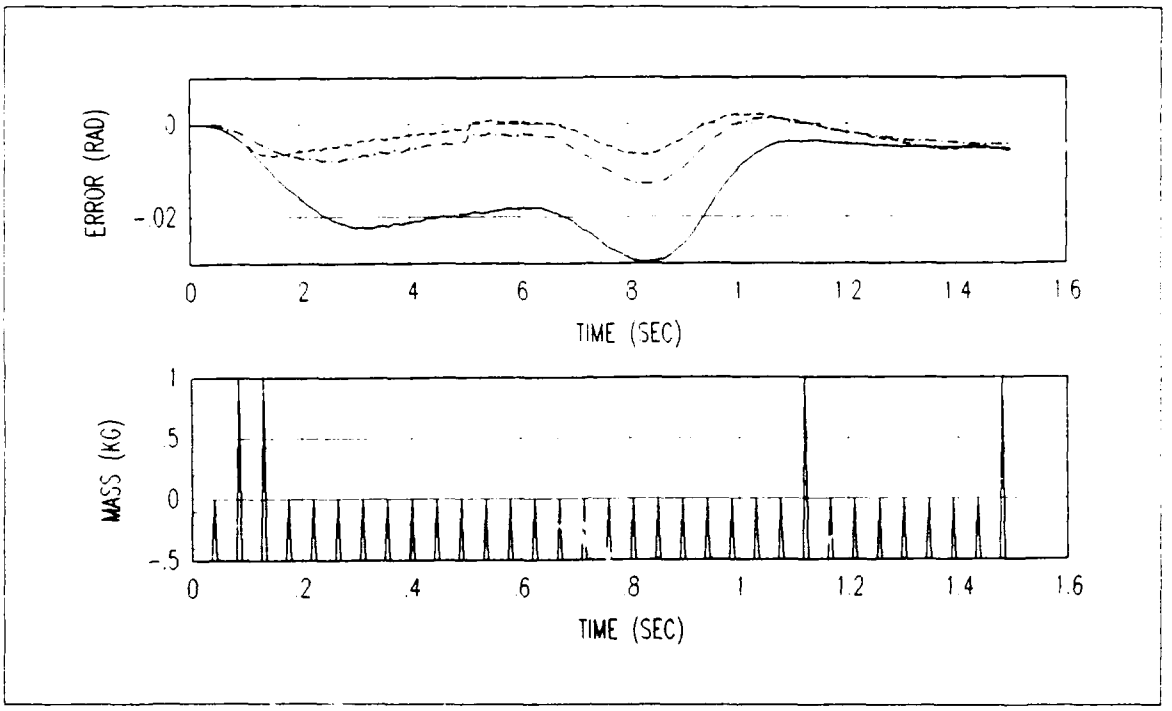

Figure 4.30. Firing of Neural Networks during Trajectory 1A with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
-·-	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs

Several explanations for the neural network behavior are possible. One explanation is that the nets are giving incorrect values in an attempt to drive the trajectory tracking error to zero. Another possibility is overlapping of decision regions in the mass parameter decision space causes the nets to give incorrect estimates. An additional possibility is that the nets require more training.

Due to the exhaustive training given the nets, more training should not be required. Since incorrect values are seen to occur during sample time periods where there is little or no overlap in the position trajectory error/payload decision space,

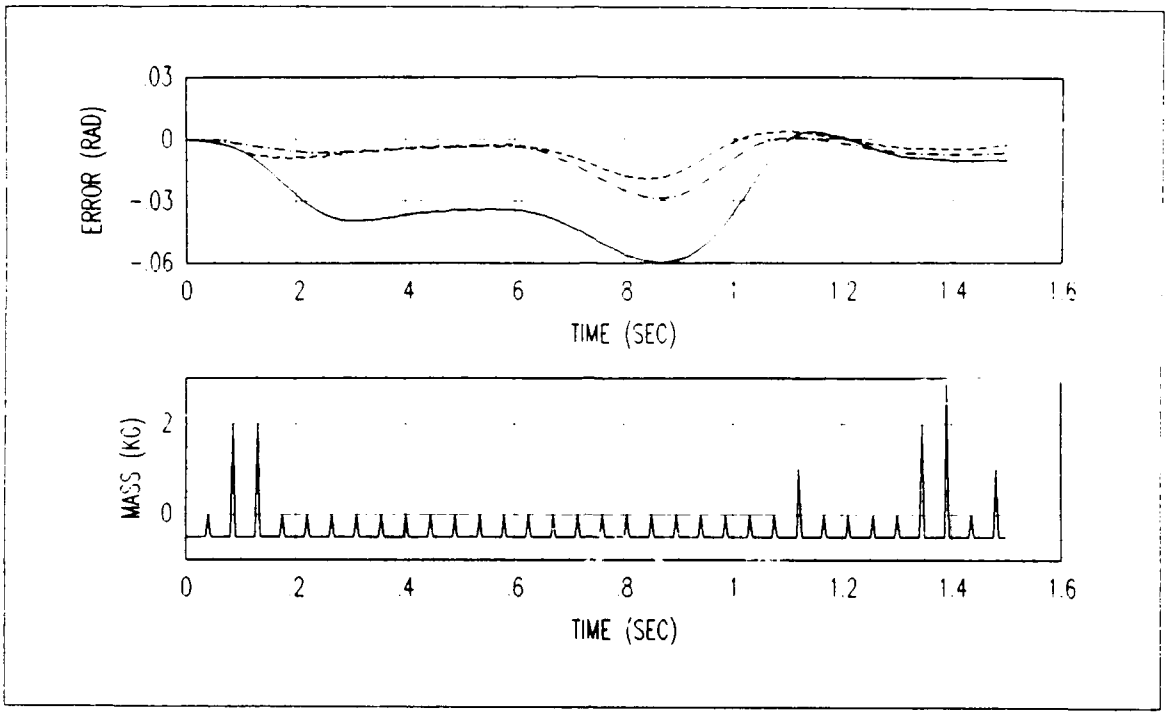

Figure 4.31. Firing of Neural Networks during Trajectory 1A with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
–·–	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs

overlapping in the de[illegible]n space cannot be the cause. Therefore, the nets must be yielding incorrect values in an attempt to drive the tracking error to zero. The endeavor of the neural nets to drive the tracking error to zero is evident in the effects of incorrect payload values on the trajectory tracking.

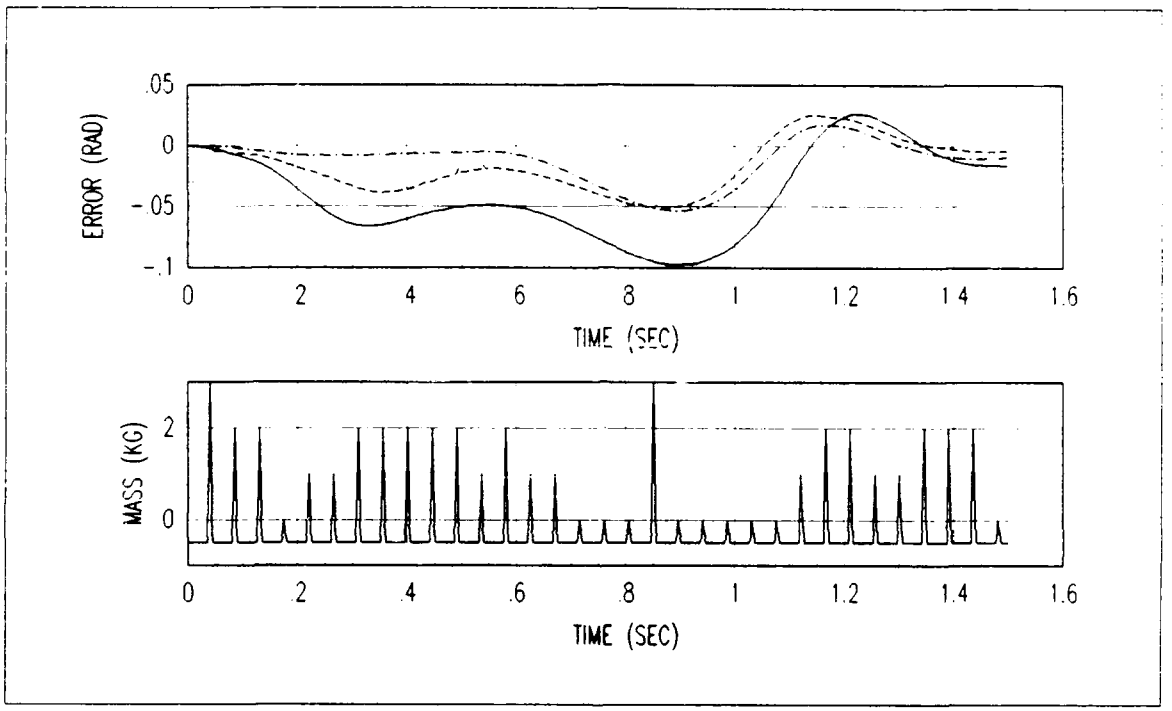

Figure 4.32. Firing of Neural Networks during Trajectory 1A with 3.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
–·–	SMBC w/3.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs (*)

4.5.7.2 Miscellaneous Trajectory Tests All of the following tests are performed on trajectories in which the neural networks are *not* trained. AMBNNC performance when correctly informed of a one kilogram payload on trajectory *1B* is shown in Figure 4.33. Throughout the entire trajectory the payload mass parameter value is changed only at 1.44 seconds of the 1.5 second trajectory. Figure 4.34 is the same trajectory with a payload of two kilograms. The AMBNNC is initially informed the payload is zero kilograms; however, the AMBNNC correctly identifies the payload by the second tracking error sampling. For trajectory *1C* with a one kilogram payload, Figure 4.35 shows the AMBNNC determining the correct payload by the second tracking error sampling. Also, note the mass parameter value of one kilogram is maintained throughout the trajectory resulting in the close following of the desired trajectory. Figure 4.36 illustrates the problem trajectory *1D* poses. Since the vertically articulated arm is essentially falling into the gravity field during this trajectory, the neural nets are not able to discern the payload until towards the end of the manipulator motion. Training the nets to indicate negative mass variations may solve the problem posed by trajectories similar to trajectory *1D*.

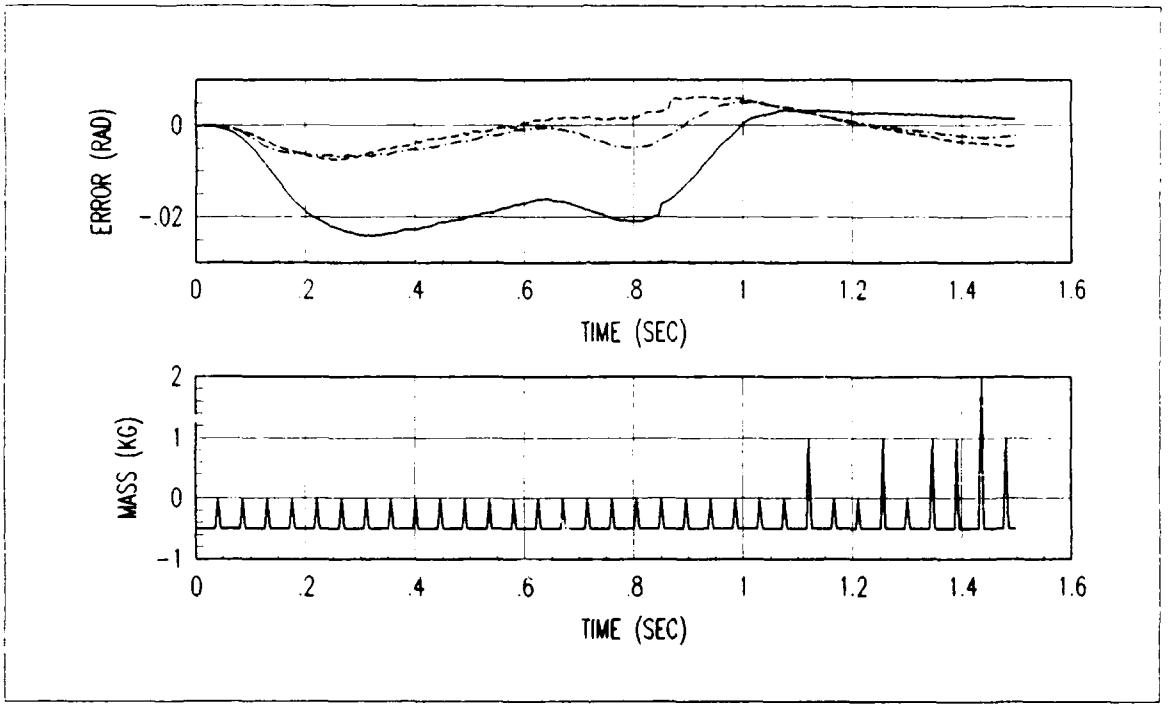

Figure 4.33. Firing of Neural Networks during Trajectory 1B with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
–·–	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/1.0 Kg Load information
∧	Neural net outputs

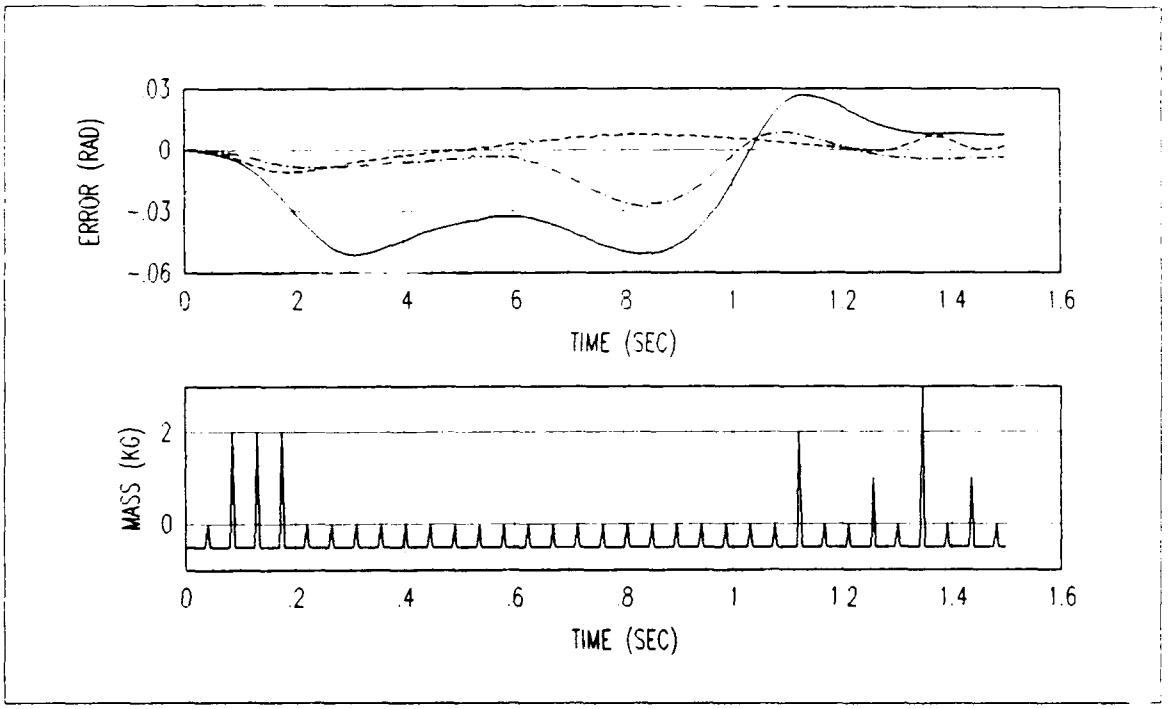

Figure 4.34. Firing of Neural Networks during Trajectory 1B with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
–·–	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs

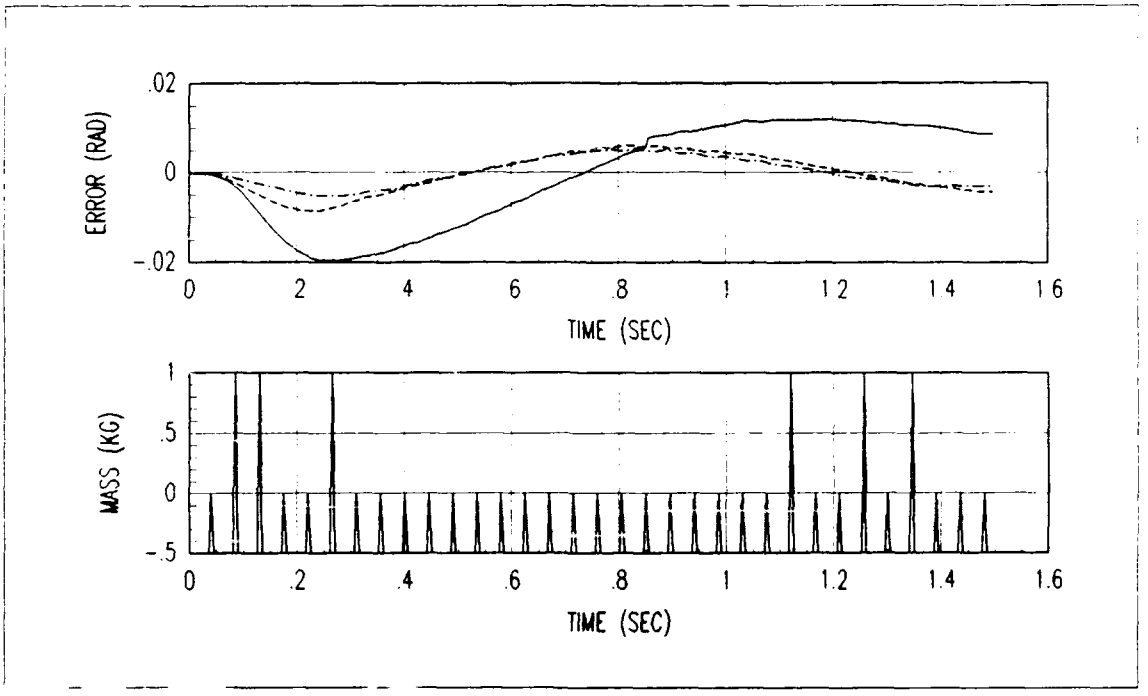

Figure 4.35. Firing of Neural Networks during Trajectory 1C with 1.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
-·-	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs

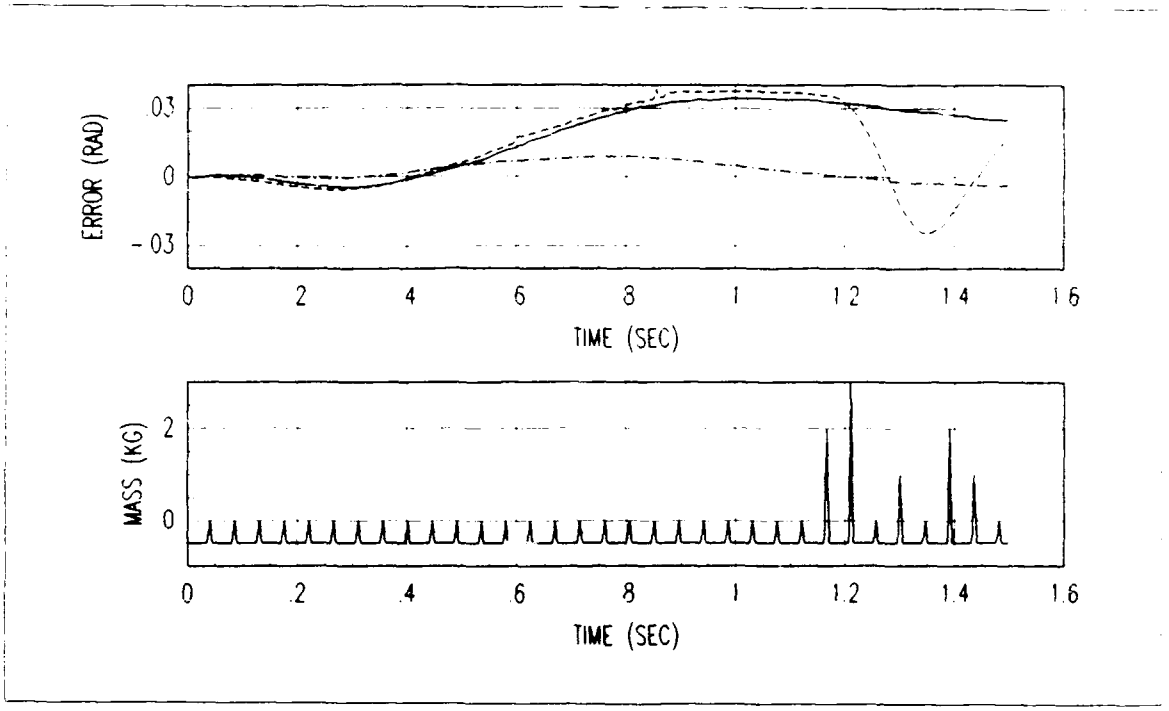

Figure 4.36. Firing of Neural Networks during Trajectory 1D with 2.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
-·-	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural net outputs

5.7.3 Negative Payload Variation Tests To further demonstrate the problem of not training the neural nets to detect negative payload variations, the following two plots depict situations in which the payload is initially misrepresented to the AMBNNC. For both Figure 4.37 and 4.38 there is no payload. In Figure 4.37 the trajectory is *1A* and the AMBNNC is initially informed the payload is two kilograms. Figure 4.38 shows execution of trajectory *1C* with the AMBNNC originally told the payload is three kilograms. Both figures indicate that the neural nets cannot give a value other than zero until a positive delta payload is detected.

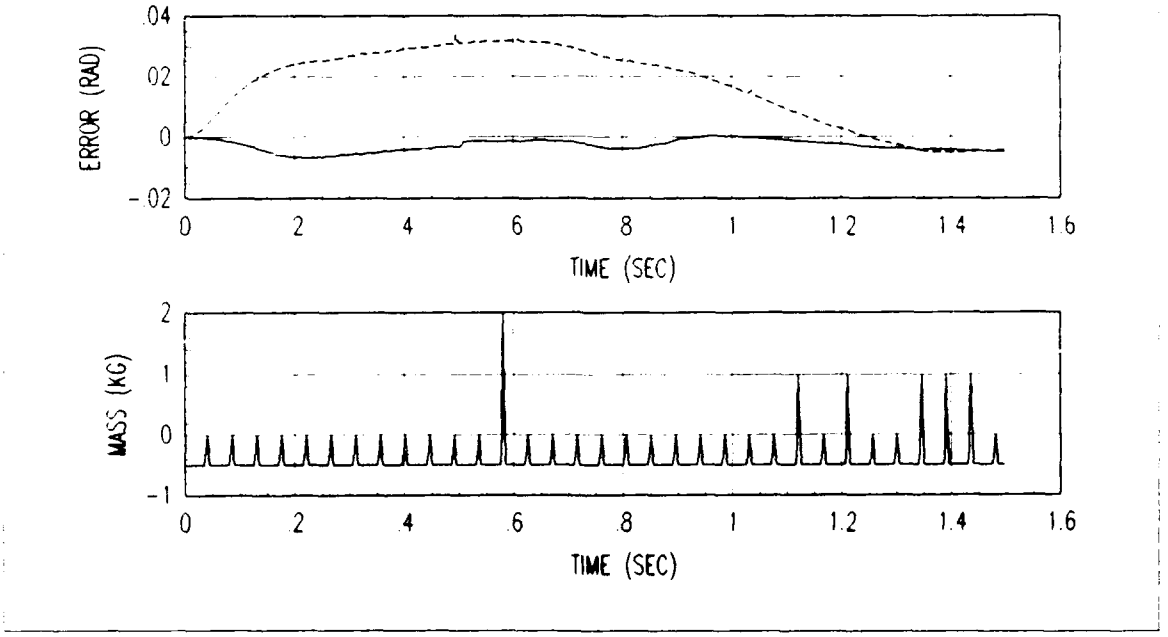

Figure 4.37. Firing of Neural Networks during Trajectory 1A with 0.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/2.0 Kg Load information
^	Neural net outputs

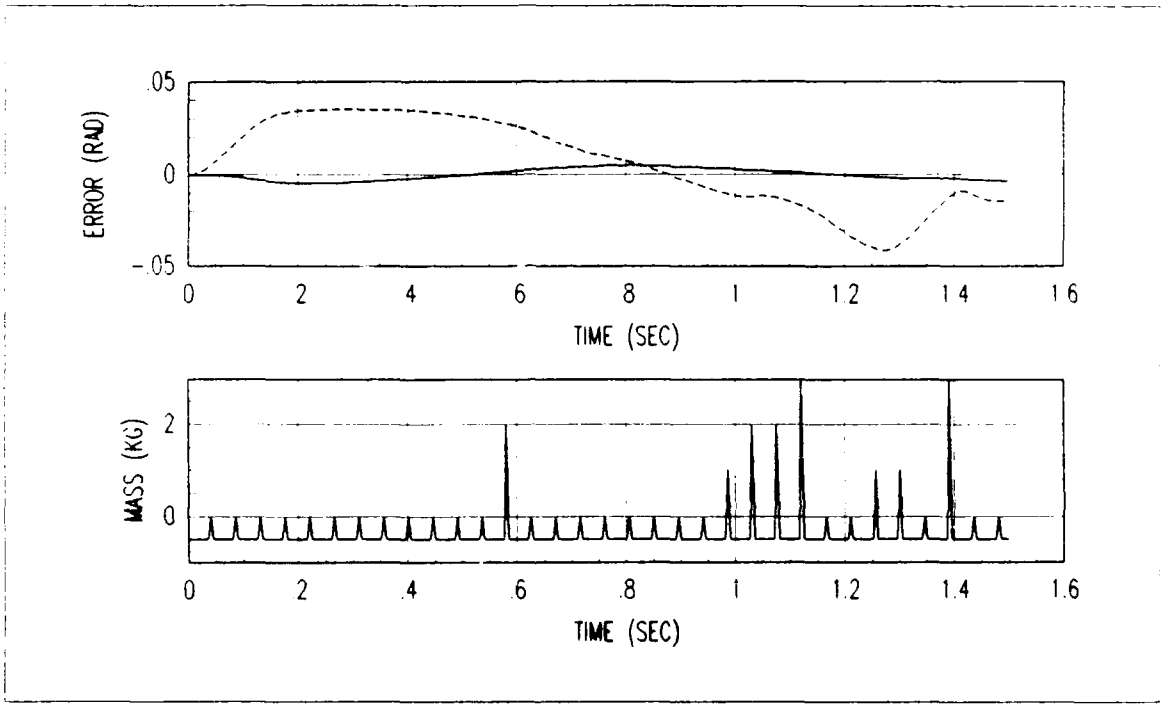

Figure 4.38. Firing of Neural Networks during Trajectory 1C with 0.0 Kilogram Payload

—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/3.0 Kg Load information
∧	Neural net outputs

4.6 Summary

The ability of neural network to identify payload mass from trajectory position error data is clearly indicated by the neural networks training with the position error data. All of the following development and validation testing defined the final form of the Neural Network Payload Estimator. The final form of the NNPE for a single PUMA joint consisted of a (2,14,14,4) structure. Final training and testing accuracy and error results decisively demonstrate that neural networks can determine the payload mass from trajectory error information.

The Adaptive Model-Based Neural Network Controller was developed to incorporate the NNPE as an adaptation mechanism. Experiments on the PUMA-560 robot authenticate the ability of the AMBNNC to identify and adapt to an unknown payload within three neural net sample periods. The AMBNNC adaptation ability is proven to extend to trajectories other than the trajectory for which the neural networks are trained. In addition, the AMBNNC performs well over a large range of payloads outside the established payload range with good convergence to a small final position error in all cases. All of these experimental results substantiate the ability of an artificial neural network to provide accurate estimates of the payload mass during high speed manipulator motion.

After the initial tests were completed, another approach to updating the payload mass parameter of the system dynamics with the NNPE estimate was tested. The new method updated the dynamics with every payload estimate. Results indicated better performance with no payload attached to the manipulator mounting flange. However, poorer performance resulted when a payload was attached. These outcomes were the opposite of those from the tests already accomplished where the dynamics were updated only when a change from a previous payload value was detected. Two combinations of both methods were tried. Using updates with every estimate up until 450 ms of a 1.5 second trajectory performed better than using updates through 225 ms. Further investigation into combining the two approaches

may provide excellent performance over the entire spectrum of payload variations. Other conclusions and recommendations are found in the next chapter.

V. Conclusions and Recommendations

5.1 Conclusions

A new concept for adaptive model-based control was proposed, developed, and experimentally validated. In the course of the development, artificial neural networks were trained to quickly and accurately estimate payload mass variations over high speed robotic manipulator trajectories. Integrating the Neural Network Payload Estimator (NNPE) into an adaptive model-based control structure provides an algorithm that produces excellent and consistent tracking performance in the presence of uncertain payloads. The tracking performance of the resulting Adaptive Model-Based Neural Network Controller (AMBNNC) was shown to be equal to or superior to a model-based controller with full prior payload information.

Investigations performed during the NNPE development indicate that 12 to 16 nodes per each of two hidden layers of a multilayer perceptron provide the best training performance with similar durations of training time. In addition, training time was found to be directly proportional to the number of training examplar vectors used, the distribution and overlapping of the trajectory error data in the mass parameter space, and the manner in which the data is normalized before being input to the neural networks. Final training, testing accuracy, and error results clearly show the NNPE capabilities to detect payload mass throughout the training trajectory. NNPE payload mass identification in both the early and late stages of the training trajectory tend to be degraded and is reflected in AMBNNC performance testing, especially during the latter stages of the trajectory.

Experiments revealed that the AMBNNC is able to provide exceptional performance over a wide range of trajectories and payload conditions. It is important to remember that the NNPEs were trained to detect payload mass variations on a single trajectory. Additional AMBNNC trajectory tracking experiments show that

neural network sample rates directly affect tracking efficiency with higher sample rates yielding better performance. The better performance at the higher sample rates is due to the nets being able to update the feedforward dynamics when a payload variation occurred. Using neural network sample rates over 40 percent lower than used during normal experimental runs, 7.2 ms versus the normal 4.5 ms, the AMBNNC still out performs the single (non-adaptive) model-based controller (SMBC).

Payload range tests revealed that for any external payload in the range tested, the trajectory tracking performance was the same for positive initial payload variances. For situations where a negative payload variation was encountered the performance was degraded over part of the trajectory, but in each case converged to a small end position error. Additionally, when the AMBNNC is initially informed of a payload within plus or minus one kilogram from the actual payload value, the peak and final position errors were superior to the SMBC performance under similar conditions. Throughout the testing regimen the trajectory execution of the AMBNNC was consistent for any given trajectory or payload condition.

The objective underlying all of AFIT's research is to seek out a method or group of methods that would enable a robotic manipulator to emulate human arm performance for Robotic Telepresence applications. One fundamental capability that must be emulated is the ability to adapt to payload variations during high speed manipulator motion. The Adaptive Model-Based Neural Network Controller has clearly demonstrated the potential to satisfy the varying payload adaptation requirement, thus bringing the Robotic Telepresence concept a step closer to realization.

5.2 Recommendations and Future Directions

Several areas of investigation could lead to an improvement in the performance already achieved with the AMBNNC. Tests showed that neural net esti-

mate sample rates affected performance. Therefore, the first potential improvement would be to use more neural nets during the trajectory execution. Only 33 nets were used to cover 334 servo sample periods in these tests. A question to answer would be, "How many and how often are payload mass estimates required for optimum performance?"

Looking for the best possible performance requires fixing several problems that caused degraded trajectory execution. One apparent fix would be to train the nets to detect negative payload mass variations to eliminate the degraded performance observed in the situations where negative variances occurred. Also, initial training with a representative sample of examplars from other trajectories could potentially lead to a version of the AMBNNC that would be payload and trajectory independent. In addition, the scheme to update the system dynamics needs to be studied.

Another area requiring examination is the discretization of the mass parameter space. One issue to keep in mind is that each division of the mass parameter space increases the size of the nets, the computational loading, the training time, and the amount of training data needed. Also, the mass parameter was considered to be a point mass in these tests. To obtain true versatility requires the capability of the AMBNNC to perform with masses where the point mass assumption is invalid. The above issues can be studied with one link motion; however, to fully explore the AMBNNC potential the investigation needs to be expanded to include other manipulator links.

During single link motion, one degree of freedom (DOF), gravity and inertia dominate the dynamic effects encountered. To excite other dynamic effects such as coriolis and centrifugal, requires investigation of more DOF. In addition, examining AMBNNC performance up through 3 DOF is required for comparison with existing experimental results using other on-line estimation methods (see [76]). Comparison with other methods of proven on-line payload estimation adaptation mechanisms is

important for deciding which method might perform better in a given application.

The multilayer perceptron structure was used primarily due to the local availability and knowledge that existed for the structure. Other neural networks exist which implement other models of neurobiological behavior. Examples are the Hopefield, Brain State In A Box, Dipole, Temporal Order Model, and Boltzmann Machine [24,63,3]. Perhaps one of these or another neural net model will give similar or better performance with increased efficiency. Perhaps a temporal multilayer perceptron model such as outlined in Appendix B might serve as a basis for further research. When talking about other types of neural networks, the issue of training becomes important. A simple gradient search was used for training the NNPEs. More efficient training methods, such as a quadratic method as suggested and used by Stright [73], exist and would decrease training time.

While several topics relevant to the AMBNNC performance potential remain to be explored, there are other areas in which the research into using the NNPE and AMBNNC techniques can be performed. Using these methods on other on-line parameter estimation problems, and using other types of manipulators are just a couple of other areas where these schemes can be tried. Sensitivity analysis of the design parameters could determine which parameters require the most attention during the design and implementation of the NNPE techniques into other control systems and environments. The AMBNNC was tested against a control law with known performance limitations to enable improvements caused by the AMBNNC to be observable. Once the AMBNNC technology is matured it can be married with more robust feedback compensation to provide operation in uncertain payload environments.

Appendix A. *Contemporary Neural Approaches to Robot Control*

A.1 Introduction

The employment of artificial neural networks (ANNs) to control robots is beginning to gain momentum, perhaps due to the excellent test bed robots provide for investigating the potential of artificial neural networks. Neural networks show potential for speech, vision, motor and sensor-motor control, tactile control, and other attributes required by robots to emulate humans [26]. To judge if neural networks can be part of an engineering solution to the design of an adaptive robot control system, three questions arise. First, what are neural networks and how do they work? Second, why use neural networks: what are their advantages over other techniques? Third, how are they being used: do previous applications relate to the application under consideration? The balance of this review provides current information for use in answering these questions.

Neural networks and how they work are covered in Chapter 2. Therefore, this discussion begins with a brief section on why neural networks are attractive for robot control. Then an extensive review of how neural networks are currently[1] being applied in robot control research is given.

A.2 Why Neural Networks?

The application of existing neural network models to various computational problems is an active area of research. The attractiveness of neural networks stems from their many inherent characteristics, including fault tolerance, the ability to process many hypotheses at the same time, and their ability to learn from and adapt to changing situations [49, page 4]. One possible specific application is parameter estimation.

[1] Originally written in May 1989, an effort has been made to note any recent experimental work.

Many of the parameters required for compliant motion control and control in uncertain environments can be determined from current sensor information using pattern behavior recognition techniques. Neural networks have been successful in pattern behavior recognition applications such as speech and vision processing [79]. If other problems can be formulated into pattern recognition terms, the use of neural networks may provide a viable solution. Additionally, all possible approaches to solving engineering and scientific processes must be looked at to determine which works best for given situations.

A.3 How are Neural Nets being used for Robot Control?

Current research into using neural networks in robotics falls into four general categories. The four categories are:

- manipulator dynamics based trajectory control,
- sensor based robot control,
- task development and control, and
- training methods.

The first category, the discussion of which is located at the end of Chapter 2, includes those studies on training neural networks the dynamic and or kinematic relationships in order to control the trajectory of the manipulator. The second category teaches the neural network to control the robot based only on sensor inputs. Sensor based robot control includes visual, tactile, proximity, other sensors, or combinations of sensor data (multiple sensor fusion). The third category focuses on those efforts to use sensor data together with task priorities and constraints to develop and control robot task strategies and performance. Task development and control includes hand grasping strategies, autonomous exploratory mechanisms, telepresence operations, etc. The fourth category is concerned with investigations into which method(s) are best suited for training neural networks used in robot control systems.

Experimental validation of developing techniques, such as training methods, is rare in current literature. Most work deals with simulation of robots, sensors, and environments. While the information garnered in this fashion is valuable for indicating possibilities, simulation is not the 'real world.' This review will focus on developments based on experimental research, when possible. However, in some areas only simulation studies are available. First, sensor based robot control is surveyed. Next, research into using neural networks to develop task strategies and control their performance is examined. Finally, a look is taken at the different training methods currently being pursued.

A.4 Sensor Based Robot Control

Marrying vision systems to robot manipulators is the most active area of neural network use in robot control. This is perhaps a direct result of neural networks' proven abilities in vision and speech pattern recognition. One simple yet representative problem many people have investigated is control of an inverted pendulum [8,12,15,79]. As shown in Figure A.1, a pendulum or "broom stick" is pinned on top of a cart. The cart travels freely along a track in the horizontal plane.

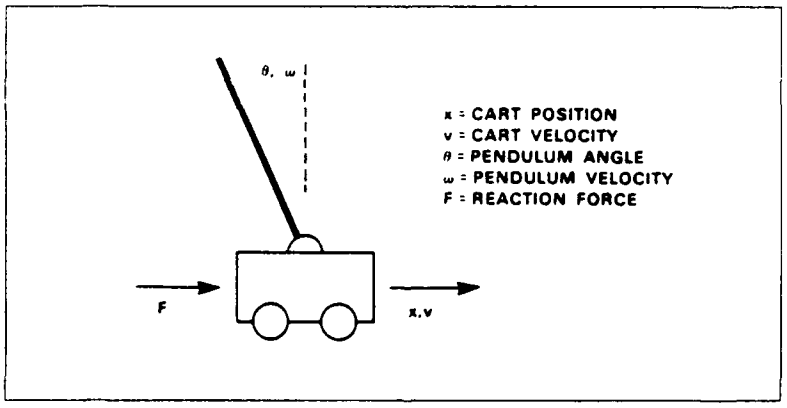

Figure A.1. The Inverted Pendulum [15]

System states are cart position and velocity, the pendulum angle with respect

to the vertical, and its angular velocity. The task is to balance the pendulum and keep the cart from hitting either end of the track. One example by Tolat and Widrow of using neural networks for this problem is in the DARPA Neural Network Study of 1988 [15]. Their control system using the states discussed above is shown in Figure A.2. Tolat and Widrow use a computer simulation for the inverted pendulum problem. One Adaptive Linear Neuron (ADALINE, see Figure A.3) is used for the neural network. The inputs to the network are two 5-by-11 quantized visual images of the cart and pendulum. Each image represents a sequential instant of time to allow the network to discern velocity information. The output of the network is the force required to stabilize the system. The system is trained using either a least mean square or Widrow-Hoff algorithm. After training, the authors state, "it was able to balance the inverted pendulum indefinitely, without crashing" [15, page 406].

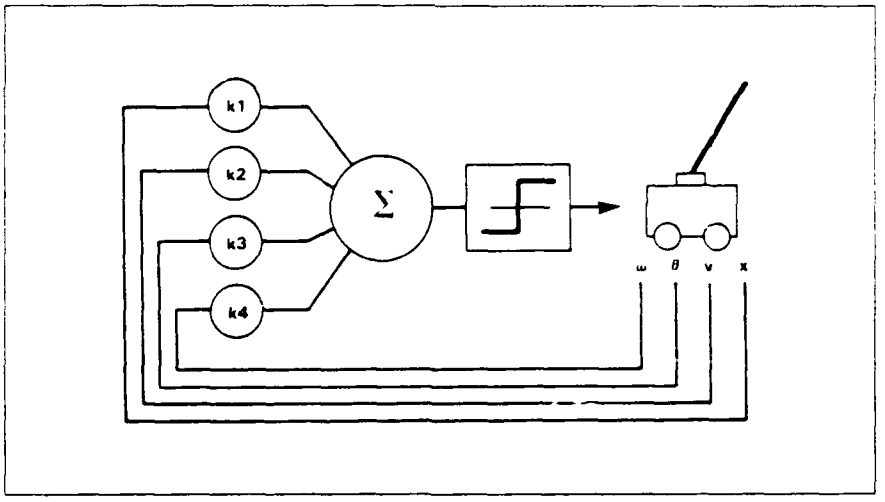

Figure A.2. An Inverted Pendulum Control System [15]

The ability of a net to learn to stabilize an unstable system shows the potential for using neural networks in similar control systems. B.W. Mel's 'MURPHY' advances the use of vision and neural nets for robot control. The following brief outline is taken from his thesis based on a "learning-by-doing" approach to robot learning [54].

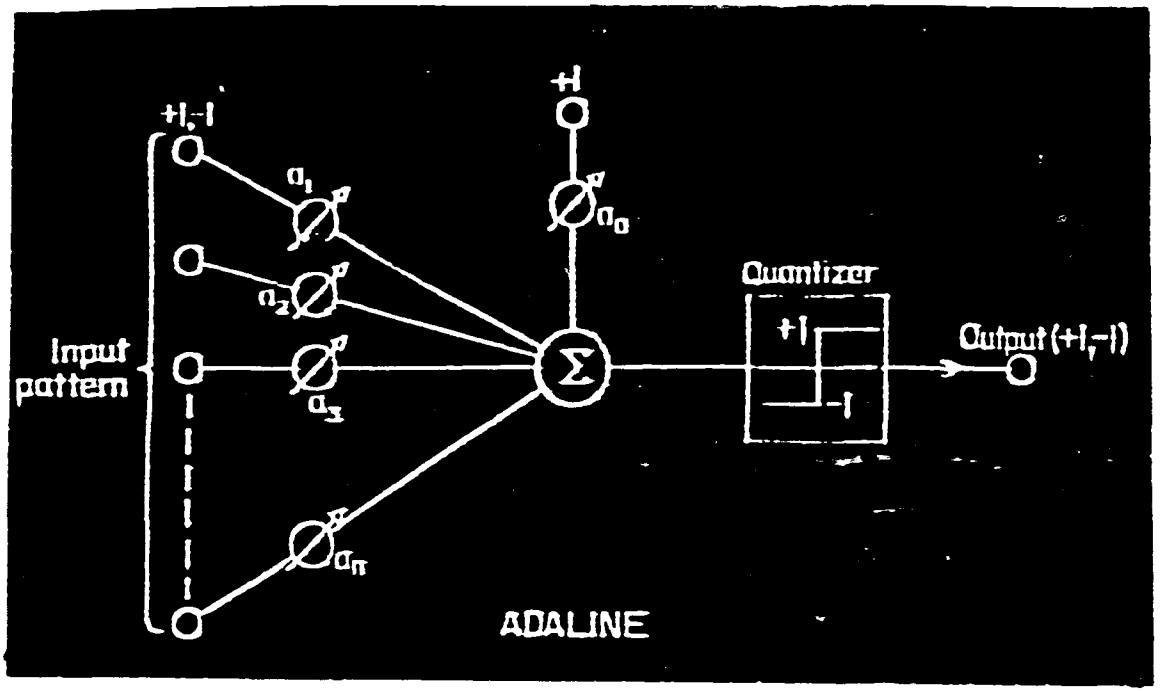

Figure A.3. An ADALINE [79]

MURPHY's basic components are four interconnected CMAC type neural networks, an autofocus camera, and a Rhino XR–3 manipulator [54, page 5]. MURPHY starts with no a priori knowledge of the arm kinematics or the vision system characteristics. Joints, manipulators, and obstacles are illuminated in white and the vision system is thresholded to 'see' only white against a dark background (see Figure A.4). MURPHY is trained, with no supervision, by showing it a representative sample of the possible arm configurations in the visual work space. After initial training, the manipulator is instructed to move from one position to another. There may be obstacles in the way and the network must decide how to reach around them to its objective. The net determines the path and the motions used by the arm to perform its task. Figure A.5, taken from the thesis, shows MURPHY negotiating obstacles to acquire a target. Experimental results show the network converges to the optimum path after several iterations. Also, the

more problems it solves, the more adept it becomes at determining the optimal path.

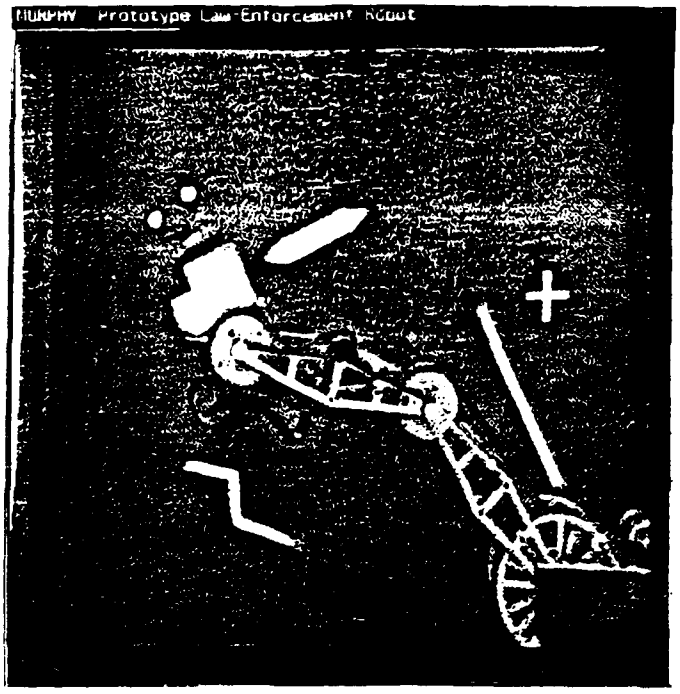

Figure A.4. MURPHY's Physical Workspace [54]

The method of approach to vision based robot control depicted with MURPHY is similar to Michael Kuperstein's approach with INFANT. However, Kuperstein uses stereooptic vision to work in a three dimensional work space. INFANT, shown in Figure A.6, stands for Interacting Networks Functioning on Adaptive Neural Topologies. Kuperstein first introduced this new architecture at the 1988 American Control Conference as a neural network designed to achieve adaptive visual-motor coordination of a multijoint robot [39]. It is designed to coordinate any number of topographic sensor inputs with any number of joint/arm configurations.

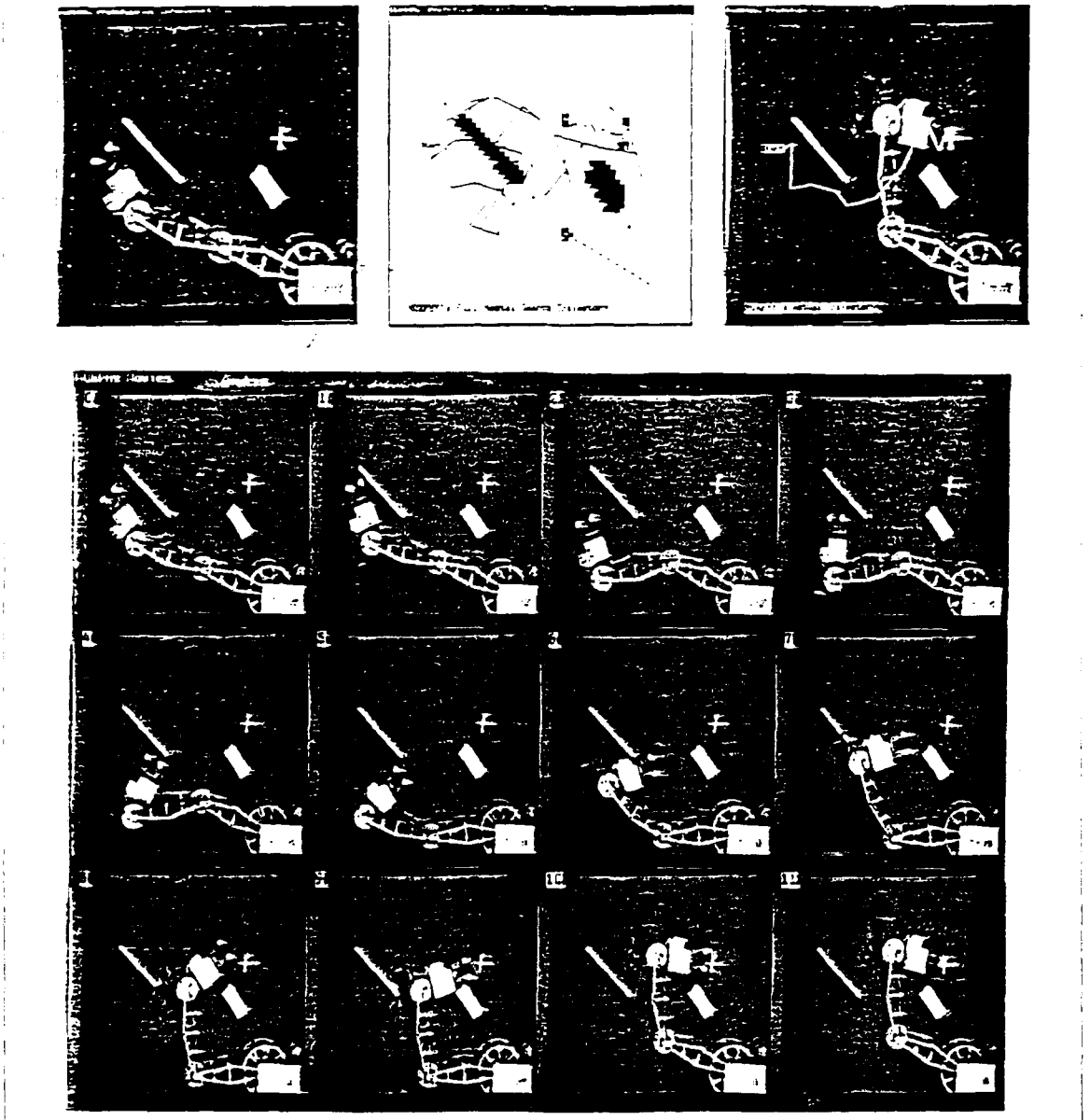

Figure A.5. MURPHY in Action [54]

The neural controller is taught using an unsupervised teaching scheme called "circular reaction" [39, page 2282]. Circular reaction is an extension of Piaget's sensorimotor stage which occurs between birth and 2 years of age. In it the "infant ...gradually becomes aware of the relationship between their own actions and their effects on the environment" [28, page 71]. Figure A.7 gives a pictorial representation of the circular reaction learning scheme.

In the scheme, sensorimotor relations are first learned through correlations between input and output signals. Then, the learned correlations are used to drive the manipulator to either reach the correct point in space or to properly grasp an object present in the visual field. Shown in Figure A.8, the system consists of an image processor, two stereo cameras, and a simple manipulator. In the initial experiments, the network learns to make the robot reach to certain points in a given work space. Recently, with a more complex version of INFANT, the network learns to "accurately grasp an elongated object without any information about the geometry of the physical sensori-motor system" [40, page 25].

Another area of active robot/sensor research is tactile perception. Tactile perception is pattern or object recognition via touch [1, page 1237]. In an article by Y. C. Pati and others, tactile perception using multiple integrated sensors is studied. A discussion taken from their article follows [59].

The authors start by listing attributes of tactile sensors. They are, as follows:

- compliant contact surfaces,
- high resolution surface stress transduction,
- local signal conditioning, and
- local computation to recover surface stress.

Figure A.9 shows a diagram of their contact surface model. The compliant contact layer is modeled as, "a homogeneous isotropic, linear elastic half-space." The strain sensors are placed below the surface a distance x.

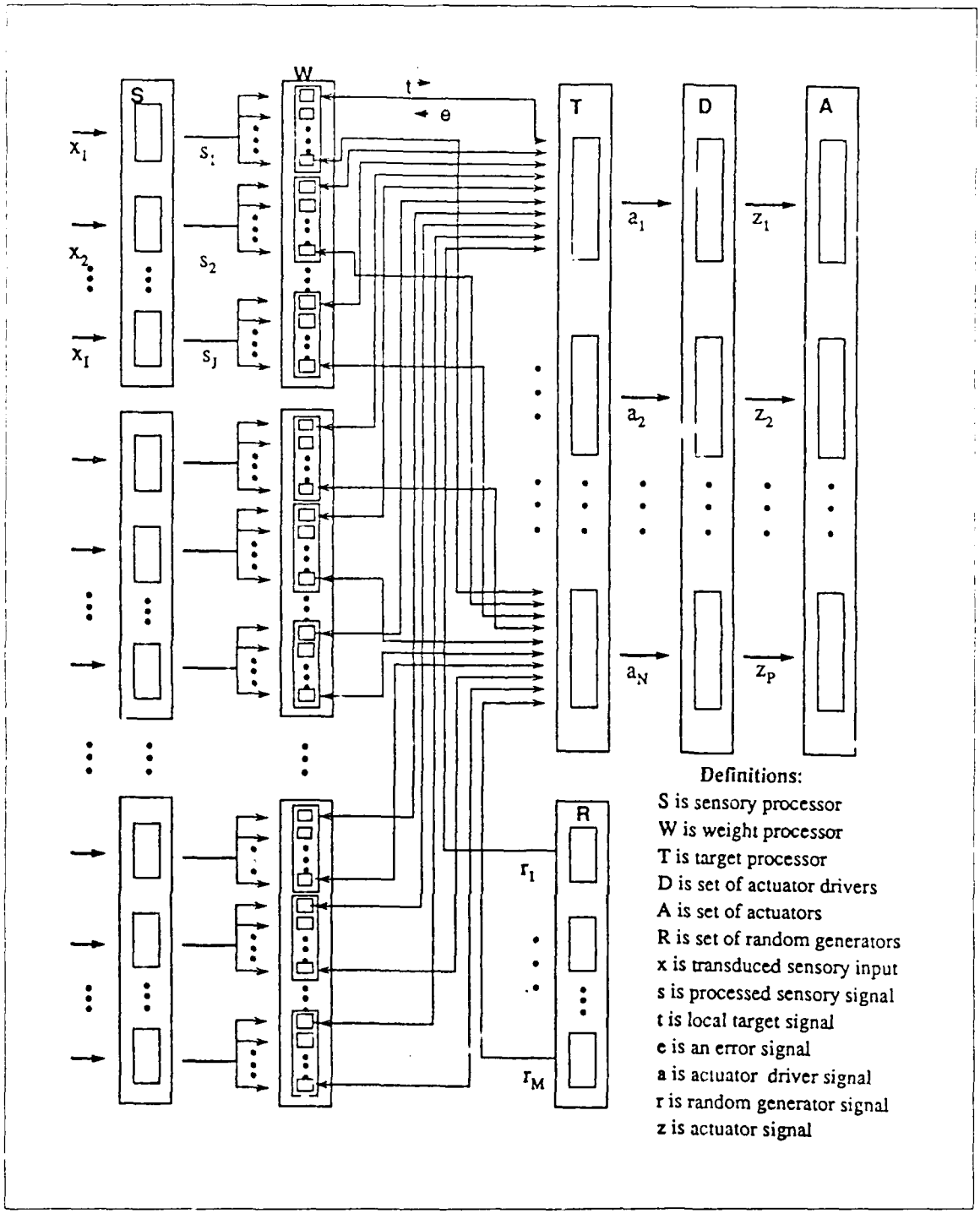

Figure A.6. A Diagram of INFANT Topography [39]

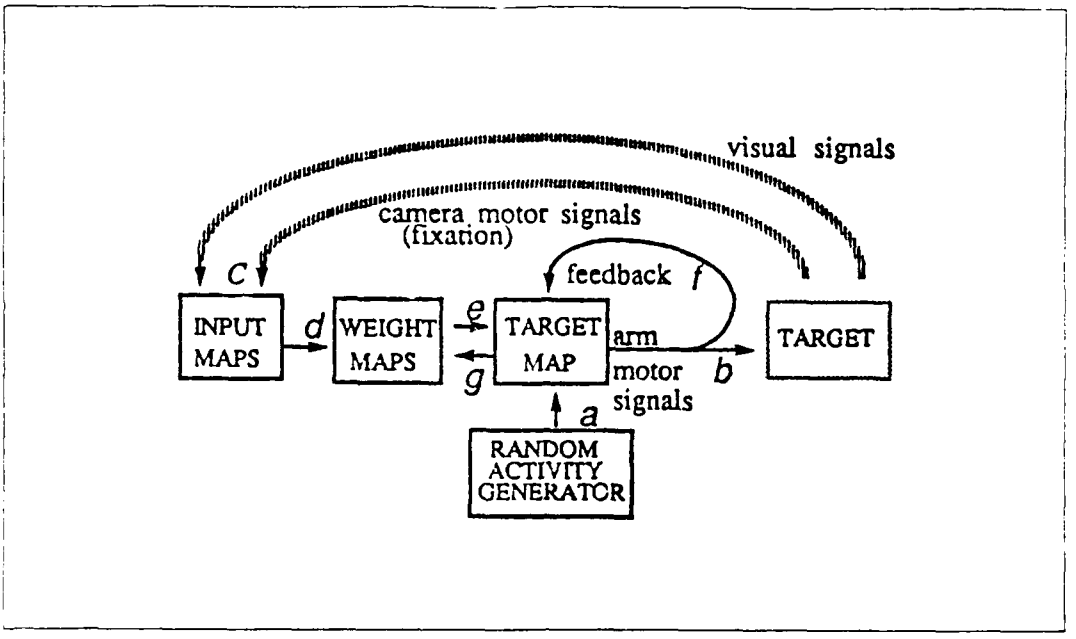

Figure A.7. Sketch of Circular Reaction Anatomy [39]

A convolution operator relates the surface stress and strains at the strain sensor depth. The 'touch' is from a cylindrical indenter which imparts its force perpendicular to the point of contact.

The neural network consists of a signal plane that determines a current "guess" of the surface stress, and a constraint plane to evaluate the guess to determine if the signal plane needs adjusting. A convolution kernel is used as an interconnection matrix between the signal and constraint planes. Exponential amplifiers are used in the signal plane. Other than these two changes, the network is essentially a Tank-Hopfield neural network [59]. The authors state their tactile system shows the ability to deconvolve applied stress profiles from strain measures in about 1 millisecond, and is able to do it in presence of noise. They also mention that their algorithm works in a breadboard version and is being made into a VLSI chip at the Naval Research Laboratory in Washington D.C.

Proximity sensors are also being employed for robot control. The Fork Lift Robot is one such use presented to the 1988 DARPA Neural Network Study ap-

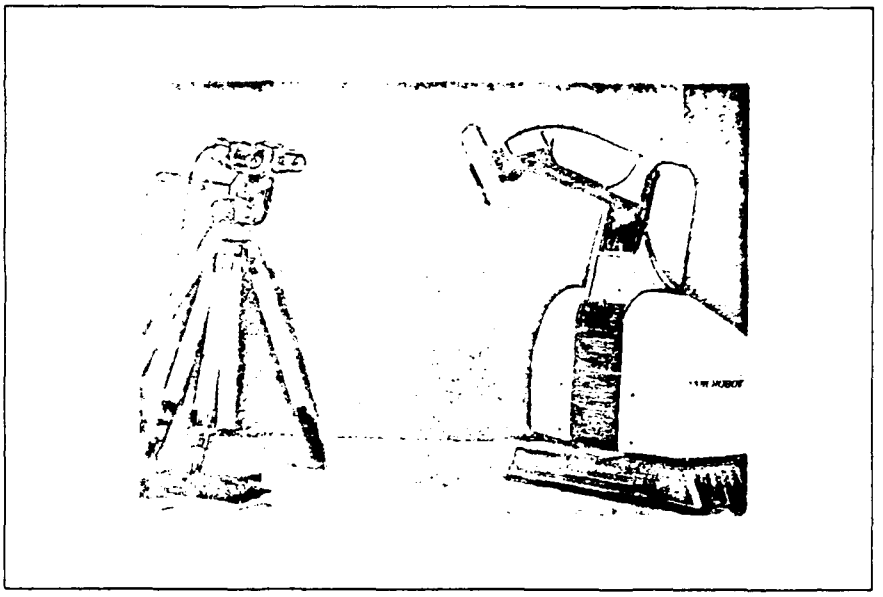

Figure A.8. Experimental Setup of INFANT [39]

plications panel. The work is done by Von Ayre Jennings [15, pages 445–450]. An industrial robot (Merlin) is fitted with a fork lift end effector. Using infrared proximity sensors, the network is taught how to acquire an offset pallet and properly insert the fork. The sensor and required motion relationships are both "extremely complicated and nonlinear" [15, page 447]. A CMAC neural network is used to generate the proper drive torques based on sensor data inputs. The network is trained in a supervised fashion by a human operator. The teacher uses a joystick placed on the robot wrist to guide the robot through the correct pallet acquisition and fork insertion maneuvers. The robot is provided with the same information the human is given during the learning process. Figure A.10 shows the project setup.

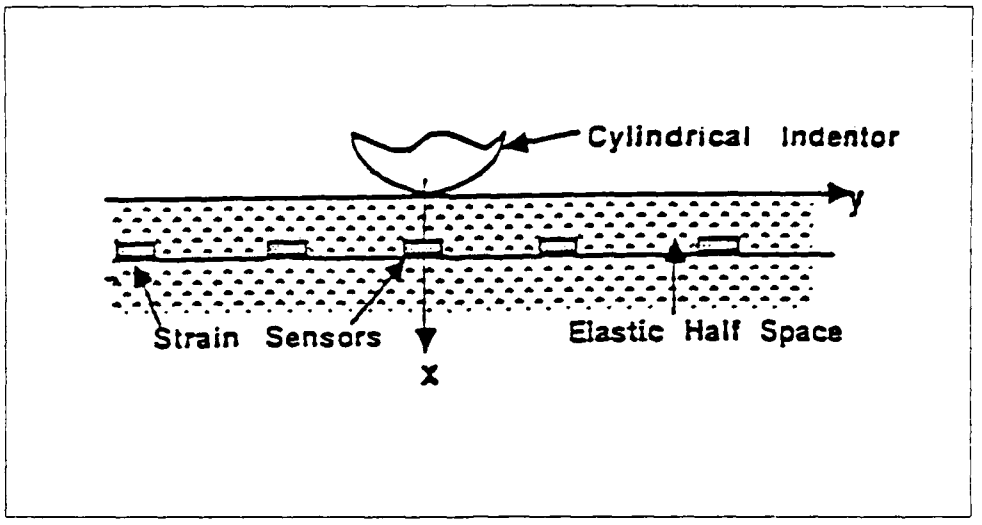

Figure A.9. Sensor Field Cross-section [59]

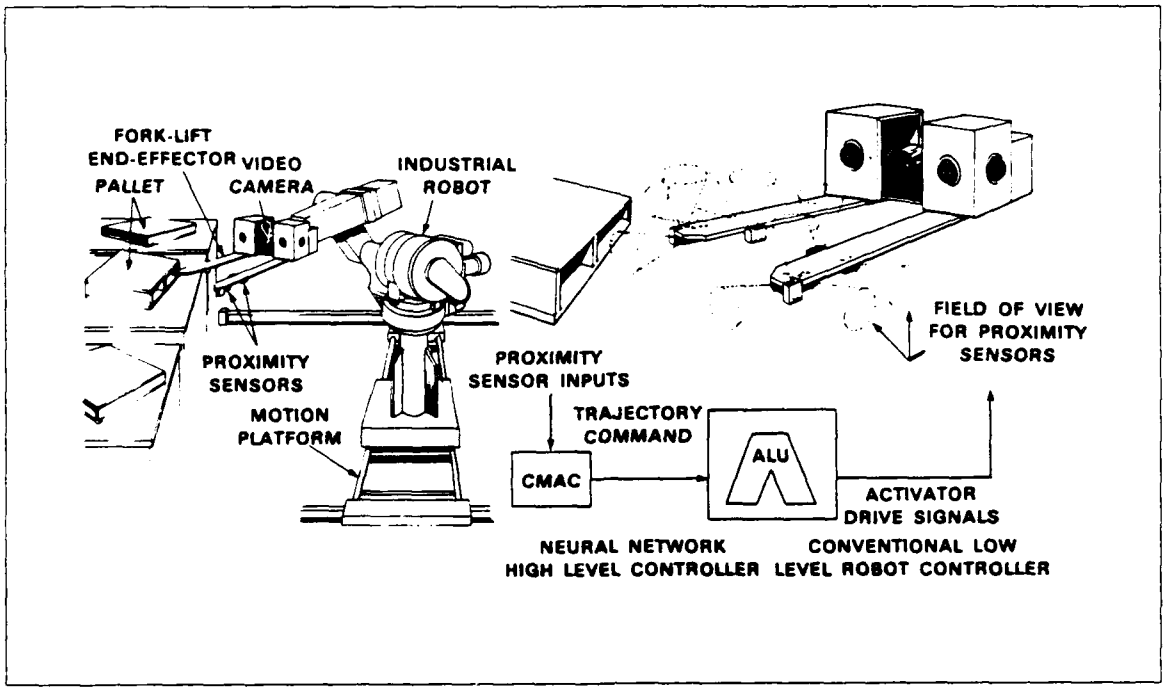

Figure A.10. Forklift Test Setup [15]

Stated experimental results indicate the neural network can perform a task autonomously after only seven teaching iterations. Furthermore, with one more training session the neural net determined the impossibility of acquiring one pallet and moved on to the next pallet. Jennings states this shows the ability of the network to, "learn complex control functions and to generalize in unexpected situations" [15, page 450].

A.5 Task Development and Control

This area brings together many of the attributes of the other four areas: sensor (visual, tactile, proximity, etcetera) data control, trajectory control, and learning. In the study of human prehension, Thea Iberall of the University of Southern California is one of the leaders. In an article presented to the American Control Conference in 1988, she discusses the use of neural networks to achieve two objectives related to robot task planning and accomplishment. They are mapping "object/task properties into prehensile posture" [31, page 2288] and determining force vectors used within a task posture to accomplish the task.

These separate but interrelated mechanisms require simultaneous resolution. One is determining the proper response posture to the task "opposition space." The other is determining the "virtual finger mapping" forces to accomplish the task. Each task mechanism uses a multilayered neural network. The inputs to each network consist of object surface length (with respect to number of finger widths), object width, amount of required force, and desired task precision. One network is trained to chose between full palm opposition or pad opposition postures. The other network is trained to determine a mapping for the individual fingers, dependent on task requirements. Training uses supervised back-propagation and a delta learning rule. The author states simulation results indicate the neural networks learned to mirror known human prehensile traits, but did not do well on tasks requiring precision. Iberall states this deficiency is probably due to insufficient

learning.

Recently; Huan Liu, Thea Iberall, and George Bekey; presented a robot hand control system based on a neural network architecture [50]. The article formalizes and further exploits the work previously done by Iberall, and represents a solid and logical progression of development of this research area. The system is called GeSAM, and consists of "a Task Analyzer and an Object Analyzer, that work together to drive the robot hand" [50, page 38]. Its basic structure is the same as discussed above but with more development into the type and size of neural network used. However, the prehensile postures differ in that they are categorized into several primitives and placed in lookup tables. Classifying the prehensile postures is an attempt at increasing the mobility of the architecture, since a different table is available depending on which of the currently developed hands it is being used on, i.e Stanford/JPL,..., Utah/MIT.

In a proposal going beyond a basic robot control system, Susan Eberlein of the Jet Propulsion Laboratory proposes the use of a hierarchical multilayered neural network to control autonomous exploratory vehicles such as the Mars Rover [17]. The net would be used as an expert decision maker employed for navigation, data transmission, and controlling scientific experiments. In this context, neural networks would be used for autonomously planning and controlling mission task accomplishment.

Along the same lines as determining the trajectory of a robot arm in the presence of obstacles, autonomous robot navigation is receiving some attention. The main thrust is to enable a robot to guide itself through terrain in new environments using only sensor data. One example is MURPHY, previously discussed, which develops a path "regardless of the presence or absence of obstacles" [54, page 46]. Another example is presented by Chuck Jorgenson in the 1988 DARPA Neural Network Study. A Hopfield neural network builds a 'world (contour) map' based on sonar data and plans a navigation path based on the map. In the map, obsta-

cles are seen as raised areas and valleys denote possible routes. The network uses the mapping to determine the optimal path to the target for the robot. Jorgenson states a demonstration unit is operational.

A.6 Training Methods

People differ on how to train neural nets. Bart Kosko defined learning as either supervised or unsupervised in a tutorial on "Associative Memory" at the 1988 International Conference on Neural Networks [36]. Supervised learning is an intelligent teacher feeding task data to a ANN until the ANN performs correctly. One example is using a teaching pendant or controller to move a robot through a given task until the robot can do it unassisted, such as with the forklift robot. Unsupervised learning is feeding input patterns into an ANN and letting it learn to detect structures within the data. Unsupervised learning is the basis of a neural network structure known as Kohonen Self-Organizing Maps, named after Tuevo Kohonen of Helsinki University [35].

Many people break the types of learning up one more step. They add a method called reinforcement learning [19, page 1096]. The premise is that every time you do something wrong you get punished, and when you do something correct you get rewarded. Sooner or later you produce correct responses for given inputs. Neural networks perform reinforcement training by feeding error signals back into the network. The error signals are the difference between the desired and actual outputs. Error information is used as a performance metric in robot control; therefore, reinforcement learning is logically suited for training neural network based robotic control systems.

Richard Elsley from the Rockwell International Science Center writes a superb article on learning architectures for back-propagation neural networks using reinforcement learning [18]. Two control architectures are covered. One involves training the ANN to be the entire system controller, which he states would re-

quire a massive neural network. The other, which he simulates, trains the ANN the inverse kinematics. A vision system is used to provide position information for comparison with the desired position inputs. Figure A.11 shows the architecture presented for training neural networks as system controllers, and Figure A.12 shows the setup for teaching inverse kinematics.

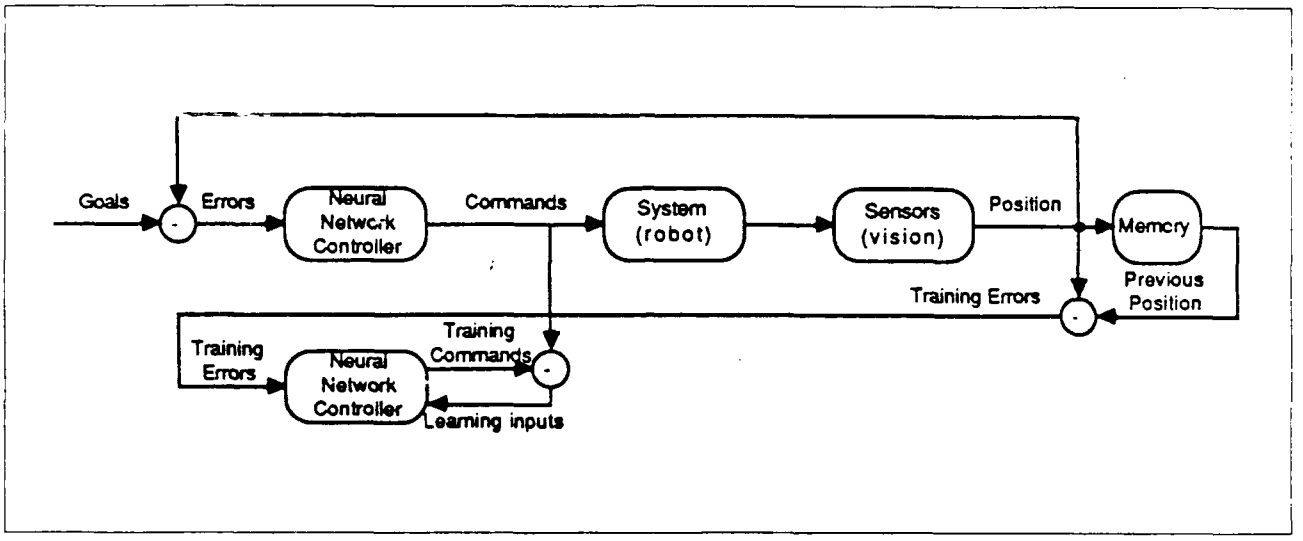

Figure A.11. System Controller Training Setup [18]

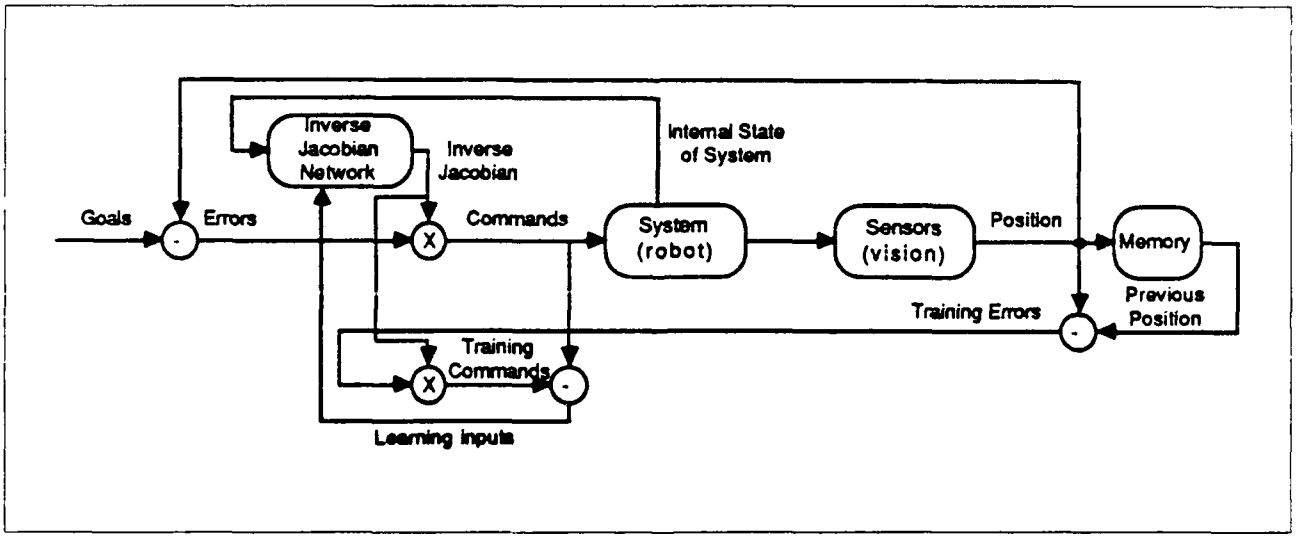

Figure A.12. Inverse Kinematics Training Setup [18]

While not specifically addressing neural networks, Judy Franklin of GTE Laboratories presents an excellent paper on reinforcement learning for robot control [19]. In the article she shows "how a system can learn about nonlinearities through experience" gained by reinforcement learning [19, page 1096]. What Franklin suggests is an arrangement that implements the processes an engineer would go through in refining an initial design for improved performance. It may be an interesting concept to explore with neural networks.

Most researchers in neural networks use the above three categories to define Neural Network learning methods. Previously discussed in the sensor based robot control section, Bartlett Mel of the University of Illinois adds another classification called "learning-by-doing" using a "sigma-pi" learning rule [54, page 16]. In this scheme, the neural network builds a mental model of the manipulator in the work space from motor and sensor inputs. When given a task, the ANN develops a solution based on relating the task to the mental model. In developing the solution, MURPHY accounts for changes (obstacles) in the work space. The abilities demonstrated by this approach show promise for use in many defense and commercial applications.

A.7 Summary

This survey has focused on current applications of neural networks to control robots and robotic devices. Brief reviews of current knowledge and abilities of ANNs started this study. The indepth examination of how neural networks are being used for robot control represents the bulk of this survey and was its primary purpose. In structuring the data, four separate yet interrelated classes emerged. For example, the article on MURPHY was summarized from different viewpoints in both the training methods and sensor based robot control sections.

Trying to find experimental evaluations of applications for each section was difficult and in some cases did not exist. In many cases only simulation research

could be found. The sparse amount of experimental data is understandable considering the recent reappearance of neural networks as an alternative computing architecture. This review reveals the need for extensive on-equipment research into how neural networks can be applied to robots and robot control.

Appendix B. *A Proposed Temporal Multilayer Perceptron*

B.1 Introduction

Multilayer perceptrons are used in static situations such as with a fixed background for vision applications, or rigorous inverse dynamics when used for a controller prefilter. However, the world is temporal. Some efforts to use neural networks for temporal applications exist. In one example, the Temporal Order Model uses time delays between nodes in both feedforward and feedback to produce short and long term memory within the same net [24,63]. Another approach uses one layer of inputs and one layer of outputs in a time sequencer scheme. One layer of weights reside between the layers and outputs are fed back as part of the next input to implement a scene pattern correlator [69]. In the thesis research, one method of adapting a static mechanism to a dynamic application has been explored. As an offshoot of the thesis work, the following development and use of a Temporal Multilayer Perceptron (TMLP) with a modified backpropagation training algorithm is submitted.

B.2 Temporal Multilayer Perceptron Operation

The arrangement and interaction of the neural network array structures presented in Chapter 3 called the Neural Network Payload Estimator (NNPE) gave the original insight into the proposed approach. The method uses weights existing between a temporal vector of neural network arrays. Terminology used in the following discussion to identify a neural net within the temporal vector of nets, is as follows:

- t_j identifies the neural net at sample period j,
- t_{j-1} identifies the neural net at sample period $j-1$, and
- $t_{j-1,j}$ identifies the connecting weights between sample periods $j-1$ and j.

The structure consists of Multilayer Perceptrons (MLPs) arranged in arrays spanning a given temporal space as shown in Figure B.1. The inputs are desired and actual joint position information, q_d and q respectively. The output is the estimated payload mass parameter $\hat{a}$.

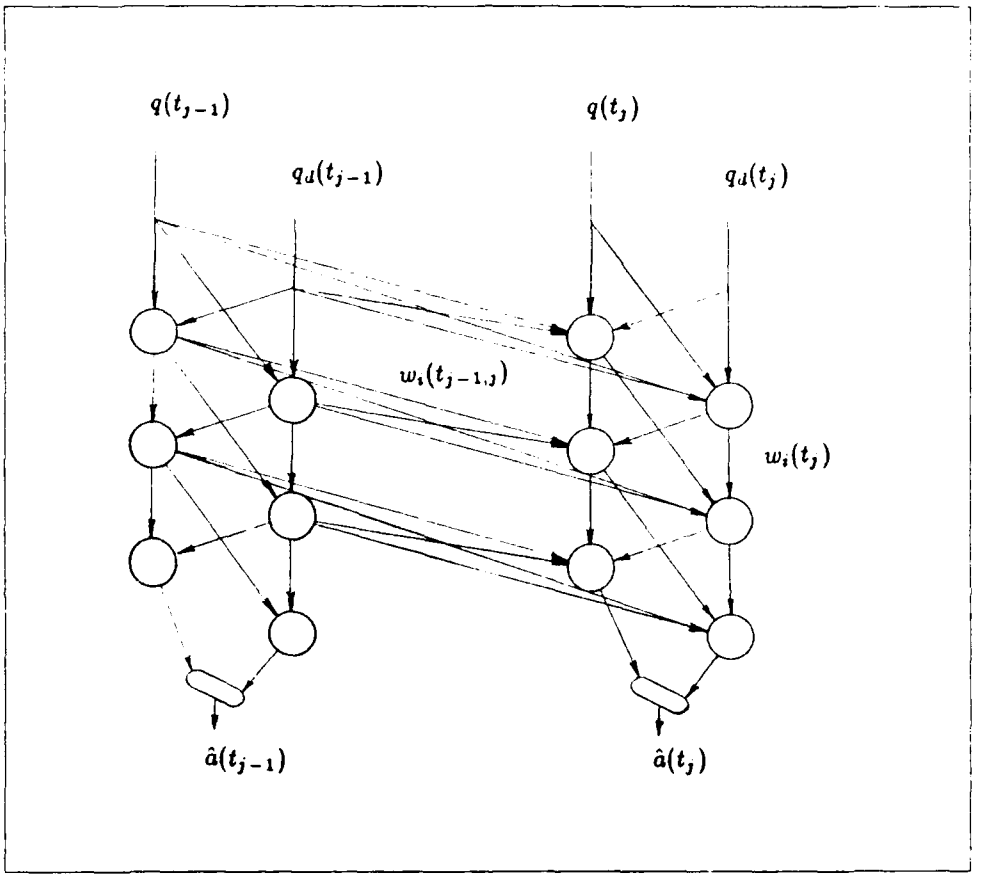

Figure B.1. Temporal Multilayer Perceptron Diagram

During feedforward operation the input to each node of the net at sample period j is governed by:

$$y(t_j) = f\left(\sum_{i=1}^{n}(w_i(t_j)x_i(t_j)) + (w_i(t_{j-1,j})x_i(t_{j-1})) - \theta\right) \qquad \text{(B.1)}$$

where:

- $y(t_j)$ is the node output in the net at sample time j,

- $x_i(t_j)$ is the ith node input in the net at sample time j,
- $w_i(t_j)$ is the ith input weight in the net at sample time j,
- $x_i(t_{j-1})$ is the ith node input from the net at sample time $j-1$,
- $w_i(t_{j-1,j})$ is the ith input weight for the inputs from the net at sample period $j-1$,
- j is the sample time,
- n is the number of inputs to the node from each net, n from the net at sample period j, and n from the net at sample period $j-1$ (a total of $2n$ inputs), and
- θ is the node threshold.

As an example each node output function $f(\bullet)$ is the sigmoidal function

$$f(\bullet) = \frac{1}{1 + e^{-(\bullet)}} \tag{B.2}$$

Except for the output nodes, the output of each node from the previous sample period neural network is weighted and used as additional inputs to the nodes in the next neural nets in the temporal sequence.

Feedforward operation of the first neural net in a temporal sequence of nets is the same as using standard backpropagation as outlined in Chapter 2. For subsequent neural nets the previous sample period net's feature vector inputs are weighted and summed along with the normal weighted inputs of the current sample time. The output of each node is the sigmoidal (see Equation (B.2)) of the sum of the weighted inputs to the node, minus the node threshold. The outputs of each of the first hidden layer nodes of the current net and the previous sample time net are weighted and applied to the current sample period neural net's second hidden layer nodes. The process just described is repeated for subsequent net layers. The output layer node outputs represent the decision of the net at the specified sample

time. However, the decision is reached using information from the previous as well as the current sample period. An additional enhancement, suggested by Dr. Steven Rogers, would be to use the previous sample period payload estimate as an additional input to current sample period neural net. The approach would enable the initial (user supplied) payload estimate to be included with the first neural network sampled. The technique may be further modified to include information from other time frames within the net currently being sampled with an additional computational penalty.

The computational overhead may be more than can be allowed. The nets have proven to give viable results when not used every sample period. Therefore, by using alternating loops, the original feedforward operations can be performed over one cycle and the between net operations can be performed during the alternate cycles. The result will be one more addition per node during the normal feedforward cycle. Before any of the above discussion has any relevance, a set of temporally trained neural nets is required.

B.3 Temporal Multilayer Perceptron Neural Network Training

To train a temporal multilayer perceptron system, using a modified Back-Propagation algorithm, the weights are initially set to random values. Next, vectors consisting of a feature vector and the desired output are presented to the network in a time sequence. The following equations illustrate the training algorithm; the implementation is discussed later.

Letting the output of each node be governed by the sigmoidal

$$f(\eta) = \frac{1}{1 + e^{-(\eta)}} \tag{B.3}$$

where η is an activation function. Deriving the equation for changing the weights is accomplished by combining two standard backpropagation equations. One equation represents the current sample period net training and the other the training

of the weights between two nets (they do not need to be identical size nets). Combining the two equations:

$$w_{i,j}(t+1) = w_{i,j}(t) + \delta_j \chi(\check{x}_i) + \alpha(w_{i,j}(t) - w_{i,j}(t-1)) \tag{B.4}$$

and

$$w_{k,j}(t+1) = w_{k,j}(t) + \delta_j \chi(\check{x}_k) + \alpha(w_{k,j}(t) - w_{k,j}(t-1)) \tag{B.5}$$

yields

$$w_{i,k,j}(t+1) = w_{i,k,j}(t) + \delta_j \chi(\check{x}_i + \check{x}_k) + \alpha(w_{i,k,j}(t) - w_{i,k,j}(t-1)) \tag{B.6}$$

where:

- $w_{i,k,j}$ is the weight between a given node, where
  - i is the upper layer's node in the current sample time net,
  - j is the next layer's node in the current sample time net,
  - k is the upper layer's node in the previous sample time net,
- χ is the training rate,
- where α is the training momentum factor,
- $\check{x}_i$ is the output of the current net's previous layer's ith node,
- δ_j is the error term for node j, and
- $\check{x}_k$ is the output of the previous net's previous layer's kth node.

The training rate χ is a number between zero and one governing the search step size. The training momentum factor α, is a number between zero and one used to prevent from being trapped in a ravine in the parameter space. The difference between the actual and desired network output becomes an error signal used to adjust the weights. For all nodes except the output nodes δ_j is computed using:

$$\delta_j = \check{x}_j(1 - \check{x}_j)\left(\sum_k \delta_k w_{j,k} + \sum_l \delta_l w_{j,l}\right) \tag{B.7}$$

where:

- $\tilde{x}_j$ is the node j output,
- $w_{j,k}$ is the weight between nodes j and k,
- $w_{j,l}$ is the weight between nodes j and l,where
  - k is the above layer's node in the current sample time net,
  - l is the above layer's node in the previous sample time net, and
  - j is the output node in the current sample time net.

For the output nodes δ_j is determined by

$$\delta_j = y_j(1 - y_j)(d_j - y_j) \tag{B.8}$$

Here y_j is the actual and d_j is the desired node j output. As each training vector is applied to the network, each connection weight is recursively updated from the output layer towards the input layer using the criteria presented above. Each node threshold is adjusted in a similar manner. When the weights and thresholds stabilize for input feature vectors representing all the classes, the network is considered trained and every weight and threshold is fixed. The previously outlined feedforward operation is used with the trained network for classifying unknown feature vector inputs in a temporal sequence.

Appendix C. *Troubleshooting*

C.1 Introduction

The following discussions are given to indicate some pitfalls and problems encountered during the thesis research effort. Hopefully, they may save others time and effort in implementing the techniques developed. The format is problem, symptoms, and solution. Also, examples of before and after behavior are given where applicable.

C.2 Problems Encountered

Problem: Neural nets failing to train.

Symptoms: After 20000 to 50000 training cycles a given neural network is not training or approaching a stable minimum.

1. Training accuracy is low and error is high and unchanging, or
2. Training accuracy and error values are not stabilizing.

Solution: Start training the given net again using a different seed. Testing with different random seed values indicated an uninvestigated correlation between certain seed values and net training aptitude.

Problem: Incorrect representation of class information in the training vectors formed during training vector formulation. To formulate training sets the algorithm uses the trajectory data filename to determine the payload class For example filename 'A11110' indicates a class of 1 kilogram. The class is noted by the '10' at the end of the filename which means a payload of 1 kilogram with the controller being told the payload is zero kilograms. The problem occurs if the filename is longer or shorter than normally used and the program keys in on the value '11' instead of '10'.

Symptoms: Each net trains to a constant low accuracy and high error. Attempts to train the nets again using different seeds do not work.

Solution: Correct class presentation in training feature vectors.

Problem: Incorrectly coded feedforward algorithm.

Symptoms: Classification testing gave results that consistently indicated the proper result for some cases and the wrong result for other cases.

1. The results were correct for about fifty percent of the inputs and when they were wrong they were only off by one payload increment. At the time assumed this to be normal for nets seeing information they had not seen before.
2. Neural net firing patterns were always the same irregardless of trajectory or payload situation.

Solution: *Changed to node threshold addition throughout the feedforward algorithm.*

Explanation: In the feedforward algorithm, the node thresholds were being subtracted as per the representation in Lippmann's article [49]. However, the algorithm used to train the nets, which was borrowed, used addition of the node thresholds.

Figure C.1 shows the neural networks firing pattern and performance with the threshold being subtracted. The same pattern was observed for varying payloads and initial conditions on trajectory *1A*. Figure C.2 show the nets firing and performance changing to threshold addition for the same payload, trajectory and initial conditions. The problem of bad nets (covered next) became apparent only after the threshold problem was corrected.

Problem: Bad individual nets.

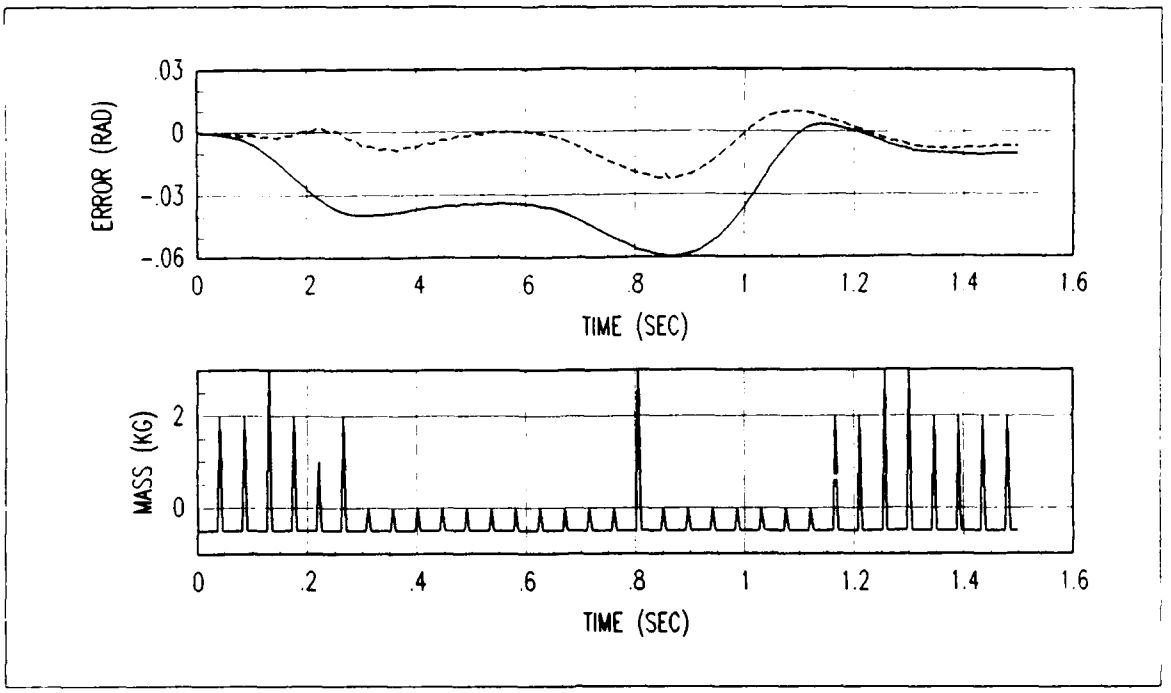

Figure C.1. Neural Net Firing Pattern when Subtracting Threshold

Payload is 2 Kilogram	
—	SMBC w/0.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information using θ subtraction
$\wedge$	Neural Net outputs

Symptoms: Some nets give incorrect results regardless of the situation.

1. Usually see the same output for every situation.
2. Sometimes the output is only off by one or two payload increments in each situation.
3. Individual net testing performance is okay.

Solution: Replace identified bad nets with newly trained nets. Sometimes retraining the nets will work. The reason for the performance problem is unknown.

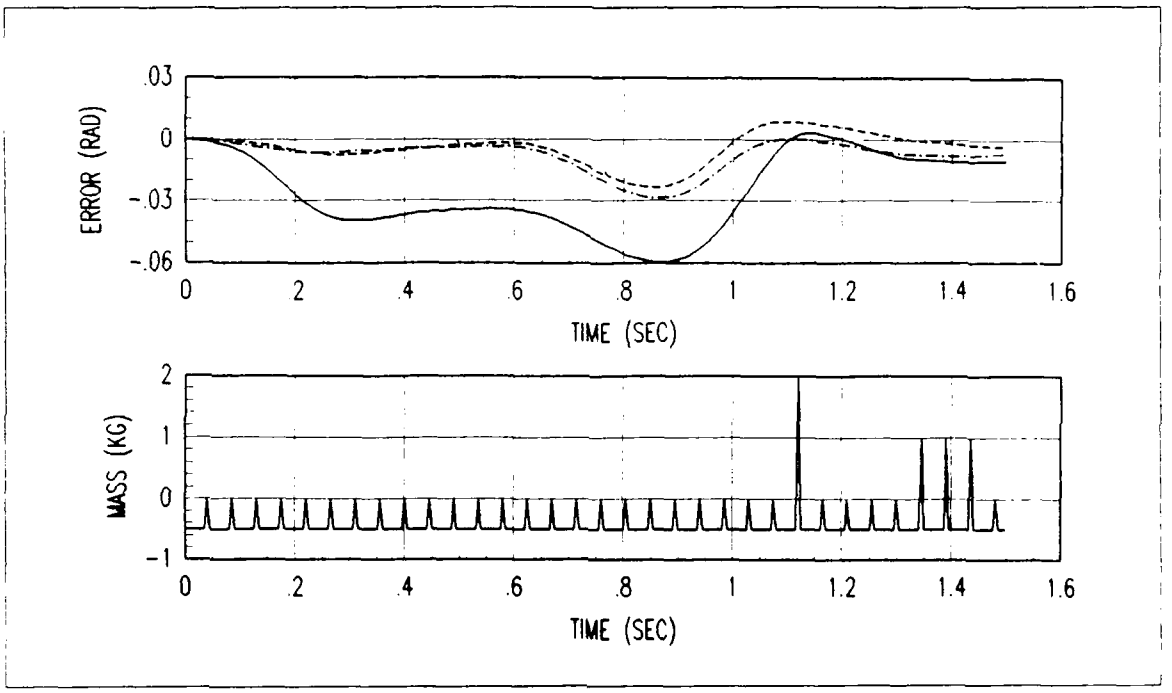

Figure C.2. Neural Net Firing Pattern after changing to Threshold Addition

Payload is 2 Kilogram	
—	SMBC w/0.0 Kg Load information
- · -	SMBC w/2.0 Kg Load information
- - -	AMBNNC w/2.0 Kg Load information using θ addition
$\wedge$	Neural Net outputs

Figures C.3 and C.4 show the nets firing pattern and performance before replacing a bad net for two different initial conditions. The bad net is at sample period 250 and is identified by giving a three kilogram payload estimate for a one kilogram payload and differing initial conditions. The same behavior is also seen in Figure C.2 for a two kilogram payload. Figure C.5 shows the firing pattern and performance after replacing the bad net. In some cases the performance was better with incorrectly firing neural nets. Two bad nets were identified and replaced with retrained nets.

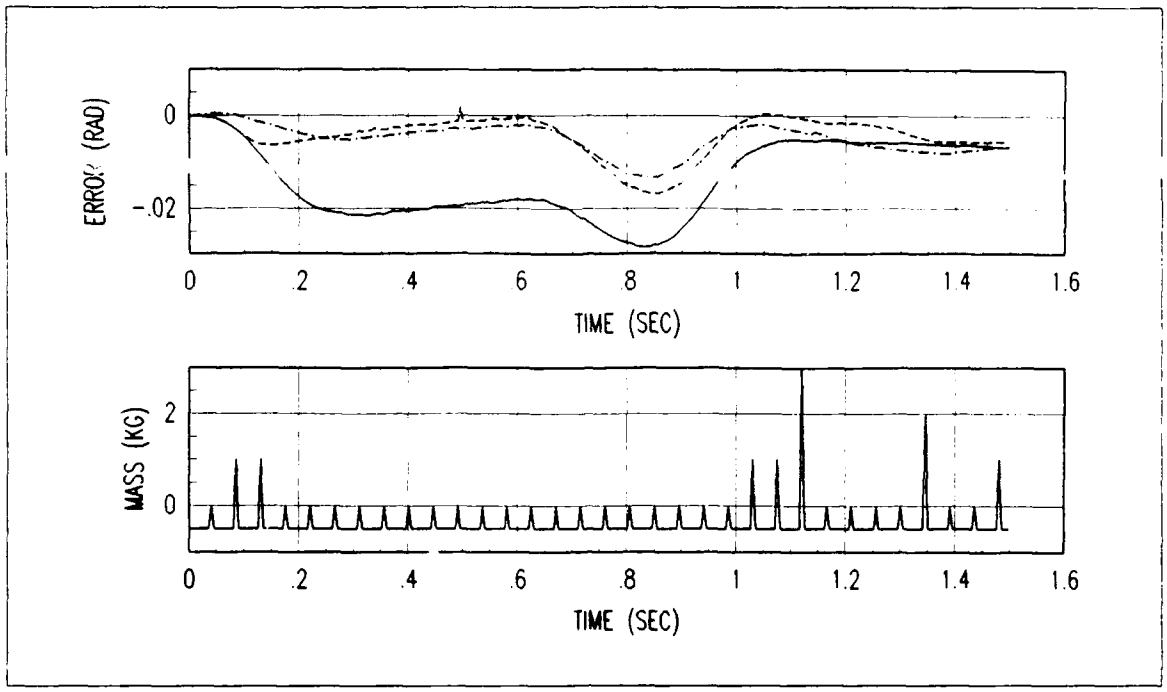

Figure C.3. Neural Net Firing Pattern showing Bad Net Firing

Payload is 1 Kilogram	
—	SMBC w/0.0 Kg Load information
- · -	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/0.0 Kg Load information
∧	Neural Net outputs

The problem given next is probably computer and/or operating system dependent.

Problem: Floating point overflow in FORTRAN math library.

Symptoms: Feedforward algorithms work one day and not the next. Investigation revealed that during feedforward execution, if the power the exponential is being raised to in the sigmoid function is greater than e^{20} the program is exited with a math over flow error.

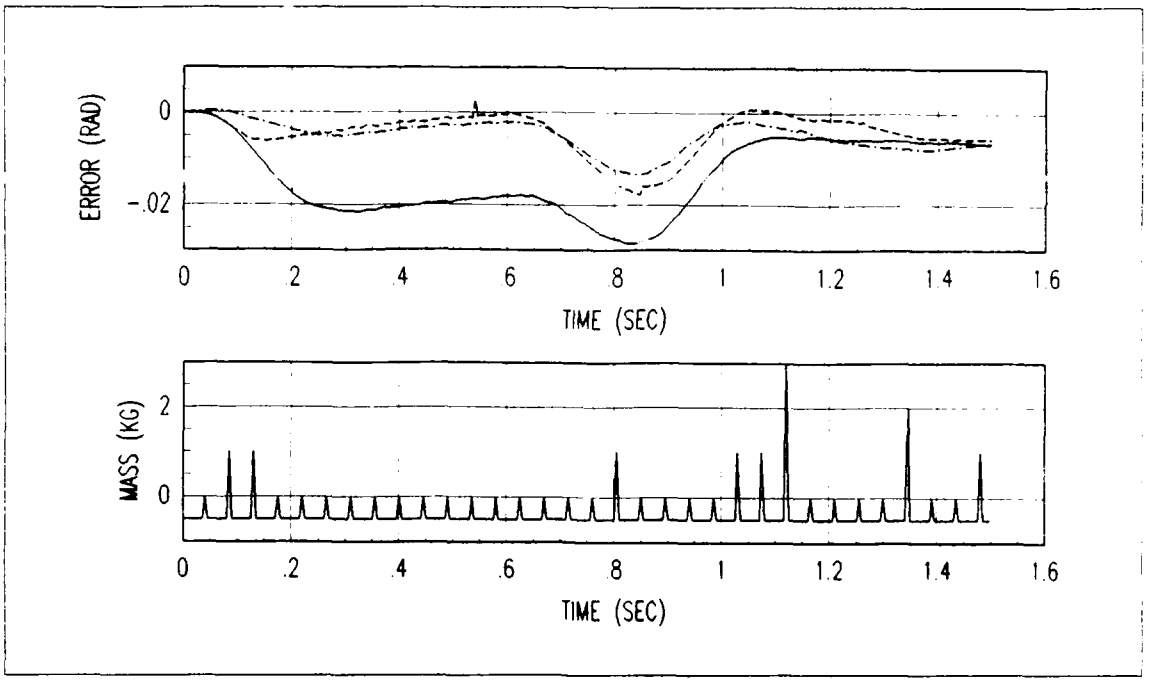

Figure C.4. Neural Net Firing Pattern showing Bad Net Firing

Payload is 1 Kilogram	
—	SMBC w/0.0 Kg Load information
– · –	SMBC w/1.0 Kg Load information
- - -	AMBNNC w/1.0 Kg Load information
∧	Neural Net outputs

Solution: The solution was to set an error trap which sets the sigmoid function to zero when the power of the exponential is greater than or equal to 20. I suspect the problem came about due to a software default being inadvertently reset.

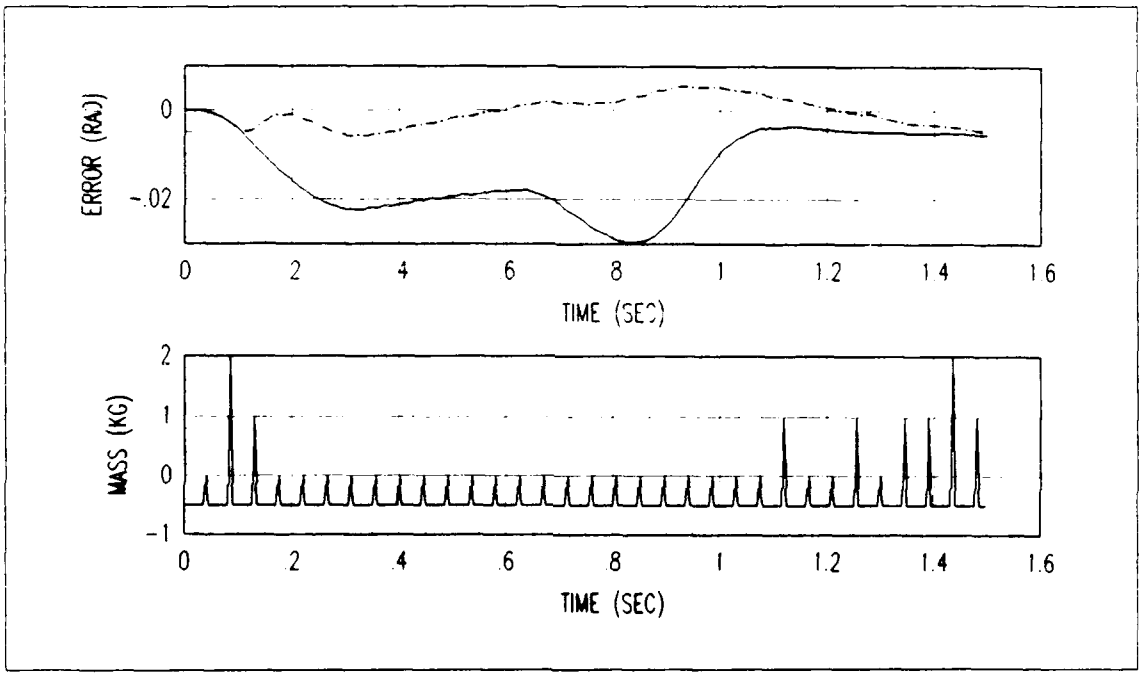

Figure C.5. Neural Net Firing Pattern after Replacing Bad Net

Payload is 1 Kilogram	
—	SMBC w/0.0 Kg Load information
- · -	AMBNNC w/0.0 Kg Load information
∧	Neural Net outputs

C.3 Summary

These were the major problems encountered during the research effort. Others that are found in future developments should be added to the list in an effort to create a comprehensive list of problems for future investigators to be wary of during implementation.

Bibliography

1. *The American Heritage Dictionary (Second College Edition).* Boston MA 1982. Houghton Mifflin Company.
2. D. L. Akin and R. M. Sanner. Neuromophic pitch attitude regulation of an underwater telerobot. In *Proc. of the 1989 American Control Conference*, pages 890-895, IEEE Press, Pittsburgh PA, June 1989.
3. I. Aleksander. A review of parallel distributed processing. In I. Aleksander, editor, *Neural Computing Architectures: the design of brain-like machines*, The MIT Press, Cambridge MA, 1989.
4. L. B. Almeida. Backpropagation in non-feedforward networks. In I. Aleksander, editor, *Neural Computing Architectures: the design of brain-like machines*, chapter 5, The MIT Press, Cambridge, Massachusetts, 1989.
5. C. H. An, C. G. Atkeson, J. D. Griffiths, and J. M. Hollerbach. Experimental evaluation of feedforward and computed torque control. In *Proc. of the 1987 Int. Conf. on Robotics and Automation*, pages 165–168, IEEE Computer Society Press, Raleigh NC, April 1987.
6. C. H. An, C. G. Atkeson, J. S. Griffiths, and J. M. Hollerbach. Experimental evaluation of feedforward and computed torque control. *IEEE Trans. on Robotics and Automation*, 5(3):368–372, June 1989.
7. C. H. An, C. G. Atkeson, and J. M. Hollerbach. Estimation of inertial parameters of rigid body links of manipulators. In *Proceedings of the 24th Conference on Decision and Control*, pages 990–995, IEEE Press, New York, December 1985.
8. C. W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9:31–37, April 1989.
9. C. G. Atkeson, C. H. An, and J. M. Hollerbach. Rigid body load identification for manipulators. In *Proceedings of the 24th Conference on Decision and Control*, pages 996–1002, IEEE Press, New York, December 1985.
10. C. G. Atkeson and D. J. Reinkensmeyer. Using associative content-addressable memories to control robots. In *IEEE Proc. of the 1988 Int. Conf. on Robotics and Automation*, IEEE Computer Society Press, Washington D. C., 1988.
11. J. Barhen et al. Neural learning of constrained nonlinear transformations. *Computer*, 22(6):67–76, June 1989.
12. A. G. Barto et al. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 2:834–846, September/October 1983.
13. P. S. Churchland and T. J. Sejnowski. Perspectives on cognitive neuroscience. *Science*, 242:741–745, November 1988.

14. Defense Advanced Research Projects Agency (DARPA). *DARPA Neural Network Study.* Executive Summary; Electronic Systems Division Contract F19628-85-C-0002, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington MA, July 1988.

15. Defense Advanced Research Projects Agency (DARPA). *DARPA Neural Network Study.* AFCEA Int. Press, Fairfax VA, November 1988.

16. M. Dvorak. *Survey of Neural Net Paradigms for Specification of Discrete Networks.* Interim report for 1987 AD-A192682, Modal Logic Corporation, Solano Beach CA, January 1988.

17. S. Eberlein. Decision making net for an autonomous roving vehicle. In *Abstract, from the 1988 Snowbird Conf. on Neural Networks for Computing*, page 333, IEEE Press, New York, 1988.

18. R. K. Elsley. A learning architecture for control based on back-propagation neural networks. In *Proc. of the 1988 Intl. Conf. on Neural Networks*, pages 587–594, IEEE Press, New York, 1988.

19. J. A. Franklin. Refinement of robot motor skills through reinforcement learning. In *Proc. of the 27th Conf on Decision and Control*, pages 1096–1101, IEEE Press, New York, 1988.

20. K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *ROBOTICS: Control, Sensing, Vision, and Intelligence.* McGraw-Hill Book Company, New York, 1987.

21. K. Goldberg and B. Pearlmutter. *Using A Neural Network to Learn the Dynamics of the CMU Direct-Drive Arm II.* Technical Report CMU-CS-88-160, Carnegie Mellon University, August 1988.

22. G. C. Goodwin and D. Q. Mayne. A parameter estimation perspective of continuous-time mode reference adaptive control. *Automatica*, 23(1), 1987.

23. G. C. Goodwin and K. S. Sin. *Adaptive Filtering, Prediction, and Control.* Prentice-Hall, Englewood Cliffs NJ, 1984.

24. S. Grossberg and G. Stone. Neural dynamics of attention switching and temporal order information in short term memory. In S. Grossberg, editor, *Neural Networks and Natural Intelligence*, The MIT Press, Cambridge MA, 1988.

25. A. Guez and Z. Ahmad. Solution to the inverse kinematics problem in robotics by neural networks. In *Proc. of the IEEE Int. Conference on Neural Networks*, IEEE Press, New York, June 1988.

26. R. Hecht-Nielson. Neurocomputing applications. July 1988. Tutorial at the IEEE Int. Conf. on Neural Networks, San Diego CA.

27. O. Hernandez and M. Das. Design of nonlinear adaptive tracking controllers for industrial robots. In *Proc. of the 1989 American Control Conference*, pages 42–47, IEEE Press, Pittsburgh PA, June 1989.

28. E. R. Hilgard. *Introduction to Psychology.* Harcourt Brace Jovanovich, Inc., New York, seventh edition, 1979.

29. B. K. P. Horn. *Robot Vision.* MIT Press, Cambridge MA, 1986.

30. B. Horne and M. Jamshidi. A connection network for robotic gripper control. In *Proc. of the 27th Conference on Decision and Control*, IEEE Press, New York, 1988.

31. T. Iberall. A neural network for planning hand shapes in human-prehension. In *Proc. of the 1988 American Control Conference*, pages 2288–2293, IEEE Press, Piscataway NJ, 1988.

32. M. A. Johnson. *Payload Invariant Control via Neural Networks: Compendium of Thesis Results and Software Support Tools.* Technical Report ARSL-89-12, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

33. P. K. Khosla and T. Kanada. Experimental evaluation of the feedforward compensation and computed-torque control schemes. In *Proc. of the American Control Conference*, pages 790–798, IEEE Press, Seattle WA, June 1986.

34. P. K. Khosla and T. Kanade. Experimental evaluation of nonlinear feedback and feedforward control schemes for manipulators. *The Intl. Journal of Robotics Research*, 7(1):18–28, February 1988.

35. T. Kohonen. An introduction to neural computing. *Neural Networks*, 1:3–16, January 1988.

36. B. Kosko. Associative memory. July 1988. Tutorial at the IEEE Int. Conf. on Neural Networks, San Diego CA.

37. L. G. Kraft and D. P. Campagna. A comparison of cmac neural network and traditional adaptive control systems. In *Proc. of the 1989 American Control Conference*, pages 884–889, IEEE Press, Pittsburgh PA, June 1989.

38. S. Y. Kung and J. N. Hwang. Neural network architectures for robotic applications. *IEEE Trans. on Robotics and Automation*, 5(5):641–657, October 1989.

39. M. Kuperstein. Implementation of an adaptive visually-guided neural controller for single postures. In *Proc. of the 1988 American Control Conference*, pages 2282–2287, IEEE Press, Piscataway NJ, 1988.

40. M. Kuperstein and J. Rubinstein. Implementation of an adaptive neural controller for sensory-motor coordination. *IEEE Control systems Magazine*, 9:25–30, April 1989.

41. M. B. Leahy Jr. Dynamics based control of vertically articulated manipulators. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pages 1046–1056, IEEE, New York, April 1988.

42. M. B. Leahy Jr. Eeng 540 class notes, robotic fundamentals. January 1988. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

43. M. B. Leahy Jr. Experimental analysis of robot control: a performance standard for the puma-560. In *Proc. of the IEEE 4th Int. Symposium on Intelligent Control*, Wright-Patterson AFB OH, September 1989.

44. M. B. Leahy Jr. Robot performance discussions. October 1989.

45. M. B. Leahy Jr. and G. N. Saridis. Compensation of industrial manipulator dynamics. *The Intl. Journal of Robotic Research*, 8(4):73–84, August 1989.

46. M. B. Leahy Jr., K. P. Valvanis, and G. N. Saridis. Evaluation of dynamic models for puma robot control. *IEEE Trans. on Robotics and Automation*, 5(2):242–244, April 1989.

47. W. Li and J.-J. E. Slotine. On-line parameter estimation for robot manipulators. In *Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics*, pages 353–356, IEEE Press, New York, August 1988.

48. P. W. Likens. *Elements of Engineering Mechanics*. McGraw-Hill Book Company, New York, 1973.

49. R. P. Lippmann. An introduction to computing with neural nets. *Acoustics, Speech and Signal Processing Magazine*, 4:4–22, April 1987.

50. H. Liu and others. Neural network architecture for robot hand control. *IEEE Control systems Magazine*, 9:38–43, April 1989.

51. P. S. Maybeck. Eeng 766 class notes and discussions, stochastic estimation and control ii. December 1989. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

52. P. S. Maybeck. Eeng 768 class notes and discussions, stochastic estimation and control iii. December 1989. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

53. P. S. Maybeck. *Stochastic Model, Estimation and Control.* Volume 1, Academic Press, Inc., New York, 1979.

54. B. W. Mel. *MURPHY: A Neurally-Inspired Connectionist Approach to Learning and Performance in Vision-Based Robot Motion Planning.* Thesis, Center for Complex Systems Research, Beckman Institute, University of Illinois, Urbana IL, February 1989.

55. R. H. Middleton and G. C. Goodwin. Adaptive computed torque control for rigid link manipulators. In *Proceedings of the 25th Conference on Decision and Control*, pages 68–73, IEEE Press, New York, December 1986.

56. T. W. Miller et al. *Real Time Dynamic Control of an Industrial Manipulator Using a Neural Network Based Learning Controller.* Technical Report, University of New Hampshire, December 1988. To appear in 1989 IEEE Journal of Robotics and Automation.

57. G. W. Neat, J. T. Wen, and H. Kuafman. Expert heirarchical adaptive control. In *Proc. of the 1989 American Control Conference*, Pittsburgh PA, June 1989.

58. C. V. Negoita and D. A. Ralescu. *Simulation, Knowledge-based Computing, and Fuzzy Statistics.* Van Nostrand Reinhold Company, New York, 1987.

59. Y. C. Pate and others. Neural networks for tacktile perception. In *IEEE Proc of the 1988 Intl. Conf. on Robotics and Automation*, pages 134–139, Computer Society Press of the IEEE, Washington D. C., 1988.

60. H. H. Poole. *Fundamentals of Robotics Engineering.* Van Nostrand Reinhold, New York, 1989.

61. F. Pourboghrat and M. R. Sayeh. Neural network learning controller for manipulators. In *Abstract, from the 1988 Snowbird Conf. on Neural Networks for Computing*, page 356, IEEE Press, New York, 1988.

62. J. S. Reed and P. A. Ioannou. Instability analysis and robust adaptive control of robotic manipulators. *IEEE Trans. on Robotics and Automation*, 5(3):381–385, June 1989.

63. R. Ricart. August 1989. Discussions concerning Dipole neural networks: their operation and application.

64. S. K. Rogers. July-November 1989. Discussions concerning neural network structures and training techniques.

65. D. W. Ruck. *Multisensor Target Detection and Classification.* Master's thesis, Air Force Institute of Technology, Air University, December 1987.

66. D. W. Ruck, S. K. Rogers, M. Kabrisky, and P. S. Maybeck. Back propagation: a degenerate kalman filter? May 1989. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH. Submitted to the IEEE transactions on Pattern Analysis and Machine Intelligence.

67. D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing, Vols 1 & 2.* The MIT Press, Cambridge MA, 1986.

68. Samuel Sablan. *MMAE Techniques for Adaptive Model-Based Robot Control.* Master's thesis, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

69. N. E. Sharkey. A pdp learning approach to natural language understanding. In I. Aleksander, editor, *Neural Computing Architectures: the design of brain-like machines*, The MIT Press, Cambridge MA, 1989.

70. J.-J. E. Slotine and W. Li. Adaptive manipulator control: a case study. In *IEEE Proc of the 1987 Intl. Conf. on Robotics and Automation*, pages 1392–1400, Computer Society Press of the IEEE, Raleigh NC, 1987.

71. C. H. Spenny. Mech 523 class notes, dynamics of robotic devices. March 1989. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

72. C. H. Spenny. Mech 723 class notes, literature study in robotics. June 1989. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH.

73. J. R. Stright. *A Neural Network Implementation of Chaotic Time Series Prediction*. Master's thesis, Air Force Institute of Technology, Air University, December 1988.

74. G. L. Tarr. *Dynamic Analysis of Feedforward Neural Networks using Simulated and Measured Data*. Master's thesis, Air Force Institute of Technology, Air University, December 1988.

75. G. Tattersall. Neural map applications. In I. Aleksander, editor, *Neural Computing Architectures: the design of brain-like machines*, chapter 4, The MIT Press, Cambridge, Massachusetts, 1989.

76. L. D. Tellman. *Multiple Model Adaptive Estimation for Robot Tracking*. Master's thesis, Air Force Institute of Technology, Air University, October 1988.

77. J. T. Tou. Software architecture of machine vision for roving robots. *Optical Engineering*, 25(3):428–435, March 1986.

78. M. W. Walker. Estimating manipulator load mass properties. In *IEEE International Symposium on Intelligent Control*, IEEE Computer Society Press, Washington D. C., 1987.

79. B. Widrow. Adaptive neural networks. July 1988. Tutorial at the IEEE Int. Conf. on Neural Networks, San Diego CA.

Vita

Captain Mark A. Johnson [redacted] in 1971 [redacted] enlisted in the USAF in December, 1971. He was trained and worked as an Electronic Communication and Cryptographic Equipment System Repairman until 1982. Including initial training, his assignments through 1982 are:

- Lackland AFB, Texas (1971 – 73),
- Incirlik AS, Turkey (1973 – 74),
- Buckley ANGB, Colorado (1974 – 76),
- Woomera AS, South Australia (1976 – 78), and
- Ramstein AB, Germany (1978 – 82).

In 1982, then TSgt Johnson, attended the University of Texas at Austin, Texas as part of the Air Force Airman Education and Commissioning Program. In 1985 he received a Bachelor of Science in Electrical Engineering from the University of Texas and went on to attend the United States Air Force Officer Training School. Commissioned in October 1985, he was assigned to Headquarters, Electronic Security Command (ESC), where he was responsible for: configuration control of ESC air and ground systems; management of command publications; evaluation of engineering proposals; development of system standards, testing methods, and evaluation criteria; and development of new system testing systems. He entered the Masters Program at the Air Force Institute of Technology, Wright-Patterson AFB, Ohio in May 1988. In October of 1989, Mark Johnson was promoted to Captain in the Regular Air Force.

[redacted]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/89D-20	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology WPAFB, OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Armstrong Aero. Medical Research Laboratory 8b. OFFICE SYMBOL (If applicable) AAMRL/BBA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, OHIO, 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
PAYLOAD INVARIANT CONTROL via NEURAL NETWORKS: DEVELOPMENT AND EXPERIMENTAL EVALUATION

12. PERSONAL AUTHOR(S)
Mark Alme Johnson, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1989, December	156

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	ROBOT, ROBOTICS, ROBOT CONTROL, ADAPTIVE CONTROL, ADAPTIVE MODEL-BASED CONTROL, NEURAL NETWORKS, PATTERN RECOGNITION, PARAMETER ESTIMATION
12	09		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: Michael B. Leahy, Captain, USAF
Assistant Professor of Electrical Engineering

Abstract on Reverse Side.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Capt. Michael B. Leahy Jr.	(513) 255-9268	AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Abstract

A new form of adaptive model-based control is proposed and experimentally evaluated. An Adaptive Model-Based Neural Network Controller (AMBNNC) uses multilayer perceptron artificial neural networks to estimate the payload during high speed manipulator motion. The payload estimate adapts the feedforward compensator to unmodeled system dynamics and payload variations. The neural nets are trained through repetitive training on trajectory tracking error data. The AMBNNC is experimentally evaluated on the third link of a PUMA-560 manipulator. Tracking performance is evaluated for a wide range of payload and trajectory conditions and compared to a non-adaptive model-based controller. The superior tracking accuracy of the AMBNNC demonstrates the potential of the proposed technique.